DIPLOMARBEIT

Titel der Diplomarbeit

„Interaktion mit Geodaten an einem Multitouch-fähigen taktilen Hyperglobus“

Verfasser

Michael Holzapfel

angestrebter akademischer Grad

Magister der Naturwissenschaften (Mag.rer.nat.)

Wien, im Oktober 2012

Studienkennzahl lt. Studienblatt:	A 455
Studienrichtung lt. Studienblatt:	Diplomstudium Kartographie und Geoinformation
Betreuer:	Ass.-Prof. Dr. Andreas Riedl

Danksagung

Jetzt, da diese Arbeit kurz vor der Fertigstellung und Abgabe steht, der Stress der letzten Tage, Wochen und Monate sich langsam in Vorfreude auf einen neuen Lebensabschnitt verwandelt, ist die Zeit gekommen sich darüber klar zu werden wie es soweit kommen konnte und zu allen Personen Danke zu sagen, die mir während meines nicht zu kurzen Studentenlebens zur Seite standen.

Als erstes möchte und muss ich mich bei meinen Eltern bedanken, die all die Jahre immer bedingungslos hinter mir standen und mich vorbehaltlos unterstützt haben. Nur dadurch wurde es mir möglich die Mühlen der Universitäten in meinem eigenem Tempo zu durchlaufen und neben einer Ausbildung (hoffentlich) auch etwas Bildung zu erwerben.

An zweiter Stelle soll hier mein Betreuer Dr. Andreas Riedl erwähnt werden, der mir schon früh die Möglichkeit geboten hat mich aktiv am Institut und in seiner Forschungsgruppe zu beteiligen und mir nicht nur bei der Erstellung dieser Diplomarbeit alle denkbare Unterstützung und Freiheiten gewährt hat.

Da ich für einen Kartographen ein ungewöhnlich technisches Thema gewählt habe möchte ich mich auch bei meinem alten Freund und Schulspezl Michael Dietz für die umfangreichen Lektionen in Sachen LED-Technik (und Mathematik, Physik,...) bedanken.

Und da man ja nicht allein studiert: Ein fettes Danke für die unvergessliche Zeit an: Ramon Bauer, Christoph Fink, Lisi Gruber und alle Studien- und Arbeitskollegen in der HRG und am Institut.

Das wichtigste zum Schluß: Christina. Vielen Dank für die Unterstützung beim Verfassen und Korrigieren dieser Arbeit. Aber Im Vergleich zu dem was Du für mich in den letzten Jahren getan und auf Dich genommen hast sind das nur lächerliche, unbedeutende Kleinigkeiten. Dafür Danke!

Und jetzt viel Spaß beim lesen...

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

KURZFASSUNG

Die „Hyperglobe Research Group“ (HRG) des Instituts für Geographie und Regionalforschung an der Universität Wien forscht seit 2005 im Bereich der taktilen Hypergloben (THG). Sie hat sich zum Ziel gesetzt vor allem durch die Entwicklung von spezialisierter Software diese Nachfolger der traditionellen, analogen Globen zu Instrumenten der Wissensvermittlung für globale Sachverhalte weiter zu entwickeln (siehe [RIE-10]:8).

In dieser Diplomarbeit wird untersucht, auf welche Weise ein solcher digitaler Globus zu einem berührungsempfindlichen Eingabegerät erweitert werden kann und welche Möglichkeiten sich dadurch für das Design neuer Bedienkonzepte ergeben.

Da eine solche Erweiterung der Funktionalität Veränderungen an der Hardware eines Globus voraussetzt, werden nach einer Einführung in die unterschiedlichen Arten der Hypergloben die gängigsten Touchscreen-Technologien vorgestellt und ihre Eignung für den Einsatz an einem THG bewertet.

Im Hauptteil der Arbeit werden die Vorgehensweise und die Ergebnisse bei der Konstruktion eines Prototyps vorgestellt. Dieser basiert auf dem am Institut vorhandenen THG, der mit einem auf Infrarotlicht-basierendem System zur Erkennung von Berührungen ausgerüstet wurde. Zusätzlich wird die Entwicklung und Funktion der dafür erforderlichen Software beschrieben und das Potential dieses Bedienkonzepts untersucht.

ABSTRACT

Since 2005 the `Hyperglobe Research Group´ (HRG), located at the University of Vienna, works in the field of tactile Hyperglobes. Its primarily goal is to develop this successor to the traditional analog globes to become an instrument for imparting knowledge about global topics. To achieve this goal the HRG focuses on the development of highly specialized software ([RIE-10]:8).

This thesis examines how such a digital globe can be expanded into a touch sensitive device and what opportunities this entails for the design of new concepts of operations.

Since such an extension of the functionality requires changes to the hardware of a globe, the current touchscreen technologies are presented and rated after an introduction to the different types of Hyperglobes

The main part of the thesis depicts the procedures and results during the construction of a prototype. This prototype is based on the available Hyperglobe at the institute, which was equipped with an infrared light based system for touch detection. Moreover, the development and the functionality of the required software is described, as well as the potential of this concept of operations.

1. EINFÜHRUNG

1.1 EINLEITUNG UND MOTIVATION

Unter allen Spielarten kartographischer Darstellungsformen nimmt der Globus eine Sonderstellung ein. Während bei zweidimensionalen Abbildungen Verzerrungen immanent sind und sich bestenfalls ein Kompromiss zwischen verschiedenen Typen finden lässt (Flächentreue, Längentreue, Winkeltreue), können diese Effekte bei einer Wiedergabe der Erdgestalt durch einen Globus nahezu vollständig vermieden werden. Die bei Karten und verwandten kartographischen Ausdrucksformen notwendige Verebnung, also die Transformation des dreidimensionalen Betrachtungsobjekts in ein zweidimensionales Abbild, entfällt beim Globus, da sowohl das betrachtete Objekt als auch dessen Darstellung dreidimensionale Körper mit derselben Grundform sind.

Trotz dieses eindeutigen Vorteils des Globus gegenüber zweidimensionalen Abbildungen, war er aber dennoch seit seiner Entstehung nicht für die Allgemeinheit zugänglich und verfügbar. Vom seinem ersten Auftreten im 15. Jahrhundert bis hinein ins 19. Jahrhundert war die Produktion dieser sehr kostspielig und aufwendig. Als Folge davon besaßen lange Zeit nur sehr vermögende Personen (Klerus, Adel, Kaufleute) Globen, die sie hauptsächlich zu wissenschaftlichen Zwecken einsetzten. Später wurden sie dann gebräuchlicher und vermehrt auch zu didaktischen Zwecken verwendet [RIE-00]. Der klassische beleuchtete Globus mit der physischen und politischen Darstellung der Erde gehört, genauso wie eine Tafel und Kreide, zur stereotypen Vorstellung eines Klassenzimmers des 19. und 20. Jahrhunderts. Anhand dieses klassischen Duo-Globus wird aber ein Nachteil ersichtlich: Die Möglichkeit verschiedene thematische Inhalte auf einem einzigen Globus darzustellen ist eingeschränkt. Nur mittels einer zuschaltbaren Lichtquelle im Inneren lassen sich zwei unabhängige Ebenen verwirklichen.

Am Prinzip und Aufbau der Globen - auf einer Grundform wird mit Papier, Folie oder Farbe ein thematischer Sachverhalt illustriert - hat sich seit seiner Erfindung sehr wenig verändert.

Erst in den letzten Dekaden sind die sogenannten digitalen Globen (DG) bzw. Hypergloben entstanden. Unter diesem Sammelbegriff werden verschiedene Arten von Globen zusammengefasst, die sich in ihrem Aufbau und ihrer Funktion zum Teil sehr stark voneinander unterscheiden. Allen gemeinsam ist jedoch, dass die Darstellung von Inhalten nicht wie beim analogen Globus durch die Aufbringung von Farben, Folien oder anderen Trägermaterialien auf eine Grundform erreicht wird, sondern dass die Inhalte mit einem digitalen Verfahren auf

eine Form projiziert werden. In den meisten Fällen geschieht dies durch die Verwendung von sphärischen Displays oder der Abbildung des Globus in einer virtuellen Umgebung. Im weiteren Verlauf dieser Arbeit werden vor Allem die taktilen Hypergloben behandelt, eine spezielle Art, bei denen ein Bild auf einen realen (daher taktilen) Globenkörper projiziert wird.

Durch den Übergang vom analogen zum digitalen Verfahren der Bilderzeugung, wurde der Funktionsumfang des Globus deutlich erweitert und die Art der Benutzung und des Umgangs haben sich verändert. Die bedeutendste Neuerung ist der Wegfall der oben erläuterten Beschränkung bezüglich der Anzahl der Themen, die auf einem Globus visualisiert werden können, da die Inhalte nicht in fixierter Form auf die Kugel aufgebracht, sondern nur flüchtig projiziert werden. Infolge ist es nun möglich neben statischen Inhalten auch komplexe Animationen und Videos zu verwenden. Diese Tatsache ist ein entscheidender Fortschritt und Vorteil gegenüber der alten Globen-Generation, denn viele Thematiken, die einen globalen Bezug aufweisen, lassen sich durch eine statische Darstellung nur unzureichend oder umständlich erläutern. Man denke hierbei beispielsweise an die Verteilung der Asche in der Atmosphäre nach einem Vulkanausbruch, den Verlauf der Oberflächentemperatur des Meeres im Zuge von El Niño/La Niña Phänomenen oder den globalen Luft- und Seeverkehr. Also grundsätzlich Sachverhalte, die in ihrer Natur dynamische sind und den Ablauf von Prozessen beschreiben.

Das Bedienkonzept eines traditionellen Globus ist sehr einfach. Meist ist er um eine starre Achse drehbar und im Falle eines Duo-Globus kann die Beleuchtung ein- oder ausgeschaltet werden. Bei einem digitalen Globus ist auf Grund der zusätzlichen Funktionen auch eine Erweiterung und Anpassung der Steuerung notwendig. Der Nutzer muss die Möglichkeit haben zwischen den einzelnen Thematiken wechseln zu können, Animationen zu starten, pausieren bzw. fortzusetzen oder die Rotation des Globus zu beeinflussen. Da Hypergloben mit digitalen Methoden der Bilderzeugung arbeiten, ist ein Computer sowie dazugehöriger Software eine unabdingbare Komponente eines solchen Systems. Daher wurden bisher bei Hypergloben auch die traditionellen Eingabemethoden eines Rechners, also eine Kombination aus Maus und Tastatur bzw. einem Touchscreen verwendet. Diese Übernahme des Bedienkonzepts ist aber keine optimale Lösung für den Einsatz an einem Globus. Durch den Zwischenschritt Computer wird keine direkte Steuerung und daher auch kein direktes Feedback erreicht. Um einen Steuerbefehl einzugeben, muss sich der Nutzer auf ein anderes Gerät konzentrieren, um anschließend seinen Fokus wieder auf den Globus zu richten, die Reaktion auf den Steuerbefehl zu überprüfen und gegebenenfalls zu korrigieren. Dieses Konzept ist also wenig intuitiv und unhandlich.

1.2 ZIELSETZUNG UND FORSCHUNGSFRAGE

In dieser Arbeit sollen daher folgende Fragen untersucht werden:

- Kann ein taktiler Hyperglobus zu einem berührungsempfindlichen Eingabegerät erweitert werden?
- Auf welche Weise kann eine solche Steuerung zur Interaktion mit Geodaten eingesetzt werden?

Es handelt sich also um den Versuch die Vorteile beider Globen-Generationen, durch die Verwendung des Globenkörpers als Eingabegerät, zu vereinen: Die direkte Steuerung des analogen Globus und die zusätzlichen Funktionen der digitalen Version. Wie bei der analogen Version soll auch der digitaler Nachfolger Berührungen erkennen und darauf reagieren. Diese Methode bietet sich aus zwei Gründen an:

a) Durch die Übernahme des Steuerungskonzepts der analogen Generation von Globen wird die Funktionalität, die ursprünglich bei der Umstellung auf digitale Technik verlorengegangenen ist, wiedererlangt.

b) Durch das vermehrte Aufkommen von berührungsgesteuerten Geräten in den letzten Jahren, wie z.B. Smartphones, tragbaren Musikwiedergabegeräten, Tablet-Computern, Navigationssystemen und anderen, ist die berührungssensitive Steuerung zu einem Standard geworden. Durch die Beobachtung des Publikums bei diversen Globen-Installationen in Museen und Ausstellungen ist der Autor zu der Erkenntnis gelangt, dass eine Vielzahl der Nutzer eine solche Steuerung bereits unbewusst voraussetzen. Vor allem jüngere Personen, sogenannte „digital natives“, die an den Umgang mit den oben genannten Geräten gewöhnt sind, versuchen beim ersten Kontakt mit einem taktilen Hyperglobus diesen durch Berührung zu bedienen.

Um die Möglichkeiten und das Potential eines solchen Bedienkonzepts zu klären, wird untersucht welche Modifikationen an Hardware und Software notwendig und realisierbar sind, um einen taktilen Hyperglobus in ein berührungsempfindliches Eingabegerät umzuwandeln, und wie eine Integration dieses Konzepts in die bereits bestehende Softwareinfrastruktur aussehen könnte. Demonstriert wird dies anhand des taktilen Hyperglobus des Instituts für Geographie und Regionalforschung der Universität Wien, an dem die Hyperglobe Research Group (HRG), der der Autor angehört, seit dem Jahr 2005 forscht.

1.3 AUFBAU DER ARBEIT

Bevor allerdings auf die theoretischen Grundlagen samt der praktischen Umsetzung eingegangen werden kann, muss zuvor noch genauer auf die digitalen Globen, die verschiedenen Arten von berührungsempfindlichen Eingabemethoden und auf bereits existierende berührungsempfindliche sphärische Displays eingegangen werden.

In der Arbeit wird zunächst im zweiten Kapitel ein Überblick über die Abgrenzung und die detaillierte Beschreibung der verschiedenen Typen von Hypergloben gegeben. Im Folgenden werden die verschiedenen Prinzipien und Techniken erläutert, die im Zusammenhang mit berührungsempfindlichen, elektronischen Geräten zum Einsatz kommen. Dabei werden diese speziell auf die Tauglichkeit für sphärische Displays untersucht. Über bereits existierende berührungsempfindliche sphärische Displays sowie deren Bezug zu Globen bzw. geographischen und kartographischen Inhalten wird im vierten Kapitel diskutiert. Im fünften Kapitel folgt schließlich die Beschreibung der Theorie und der praktischen Umsetzung des skizzierten Konzepts am Hyperglobus der HRG. Diese umfasst sowohl den Aufbau und die Prinzipien der dazu notwendigen Hardware als auch der Software. Weiterführende Überlegungen, die im Rahmen dieser Arbeit nicht praktisch umgesetzt wurden, Verbesserungsvorschläge und ein Ausblick auf zukünftige Forschungsthemen in diesem Bereich werden im sechsten Kapitel besprochen. Der letzte Abschnitt enthält eine Zusammenfassung der Ergebnisse und beschließt diese Arbeit.

2. Digitale Globen

2.1 DEFINITION UND BEGRIFFSABGRENZUNG

Um den Gegenstand der Untersuchungen - die digitalen Globen - von ihren Vorgängern abzugrenzen, sollen hier zunächst die bisher gängigen Definitionen der analogen Globen betrachtet werden. Die unten angeführten Definitionen finden sich in [RIE-00].

> *„Globus (lat. Globus = Masse, Ball, Kugel), ein maßstabsgerecht verkleinertes dreidimensionales Abbild der Erde in Form einer Kugel aus Pappe, Holz, Metall oder Kunststoff, das mit einer in Merdidianstreifen zerlegten Karte beklebt ist. Der Erdglobus ist um eine durch die Pole verlaufende, um 23° 27´ gegen die Senkrechte geneigte Achse drehbar.“*
> [RIE-00]:7 nach [WIT-79]

> *„Globus (lat.: Kugel, Ball). Dreidimensionale, kartenverwandte kartographische Ausdrucksform der Erde, der scheinbaren Himmelskugel, seltener des Mondes oder eines Planeten. Sie geben gegenüber einer Weltkarte, einer Himmelskarte bzw. einer Karte eines Gestirns das Urbild unverzerrt wieder.“*
> [RIE-00]:7 nach [KRE-86]

> *„Globen sind Nachbildungen der Erde, eines anderen Weltkörpers oder der scheinbaren Himmelskugel. Sie bestehen aus Holz, Pappe, Blech, Glas oder Kunststoff und weisen meist Durchmesser von rund 25 bis 50 cm auf, was bei Erdgloben Maßstäben von 1:50 Mio. bis 1:25 Mio. entspricht.“*
> [RIE-00]:7 nach [HAK-94]

Die Definitionen nach WITT und HAKE sind für die Beschreibung digitaler Globen nicht dienlich, da sie sich durch die explizite Erwähnung von Materialien, eines speziellen Herstellungsverfahrens und durch die Festlegung auf einen bestimmten Maßstabsbereich direkt auf analoge Globen beziehen.

Einzig die Definition nach KRETSCHMER schließt einen digitalen Globus nicht aus. Allerdings liefert RIEDL in [RIE-00] eine allgemeinere und verständlichere Begriffsbestimmung, die allen

Arten von Globen gerecht wird:

> *„Ein Globus präsentiert ein maß*[stabs]*gebundenes und strukturiertes Modell eines Himmelskörpers (bzw. der scheinbaren Himmelskugel) in seiner unverzerrten dreidimensionalen Ganzheit.“*
> [RIE-00]:8

Zusätzlich führt RIEDL den Begriff des Hyperglobus ein. Die Vorsilbe „hyper“ bezieht er dabei auf ihre Verwendung im Kontext von Hyperlink, Hypertext und Hypermedia. Also der Verknüpfung von Inhalten, seien es Texte oder Textstellen, Bilder, Audiodokumente oder andere Arten von Medien. Denn ähnlich eines Hyperlinks innerhalb einer Website lassen sich bei einem Hyperglobus ebenso die dargestellten Inhalte untereinander verbinden und in einen Zusammenhang bringen.

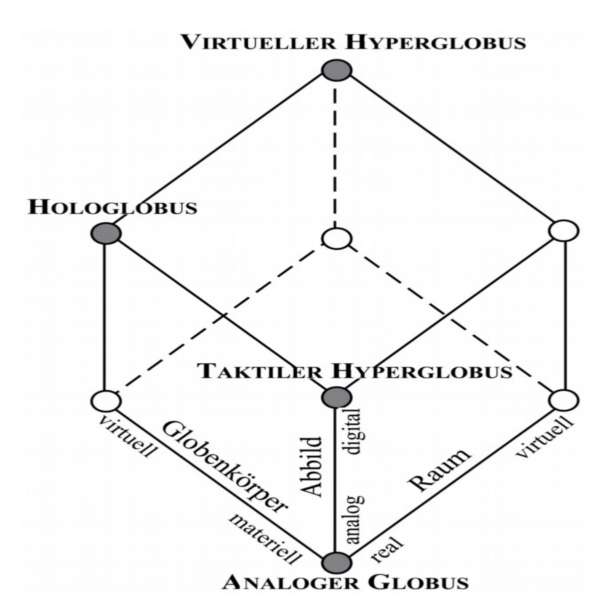


Abbildung 1: Globen differenziert nach deren Beschaffenheit (Quelle: [RIE-00]:9)

Um die verschiedenen Typen von Globen zu trennen, verortet RIEDL sie in einem dreidimensionalen gedanklichen Konstrukt (siehe Abbildung 1). Die drei Achsen stehen für die Grundkomponenten eines Globus, den Globenkörper, das Abbild und den Raum, in dem sich der Globus befindet. Jede der drei Dimensionen weist eine binäre Ausprägung auf: „virtuell“ und

„materiell“ für den Globenkörper, „real“ und „virtuell“ für den Raum sowie „digital“ und „analog“ für das Abbild.

In diesem Konstrukt lassen sich nun alle bisher bekannten Globen eindeutig beschreiben und die Hypergloben folgendermaßen unterscheiden:

- Hologloben:
  Ein digitales Abbild auf einem virtuellen Globenkörper im realen Raum

- virtuelle Hypergloben:
  Ein digitales Abbild auf einem virtuellen Globenkörper im virtuellen Raum

- taktile Hypergloben:
  Ein digitales Abbild auf einem materiellen Globenkörper im realen Raum

Der klassische, analoge Globus ist demnach eine analoge Abbildung auf einen materiellen Globenkörper im realen Raum und differiert lediglich in der Art der Abbildung vom taktilen Hyperglobus.

Zur Beurteilung der Vor- und Nachteile der verschiedenen Globentypen erstellt und verwendet RIEDL in [RIE-00] sogenannte Globenprofile. Ein solches Profil stellt eine Matrix dar, in der ein Globus anhand von zehn verschiedenen Parametern klassifiziert wird. Diese Parameter sind:

- Treueeigenschaften
- Didaktikeignung
- Interaktivität
- Themenauswahl
- Maßstabsverfügbarkeit
- Transportfähigkeit
- Produktionsaufwand
- Aktualität
- Bedienbarkeit
- Repräsentativität

Anhand solcher Profile ist es möglich die verschiedenen Typen schnell und einfach zu vergleichen. Abbildung 2 zeigt die Profile eines analogen Globus und eines taktilen Hyperglobus im Vergleich.

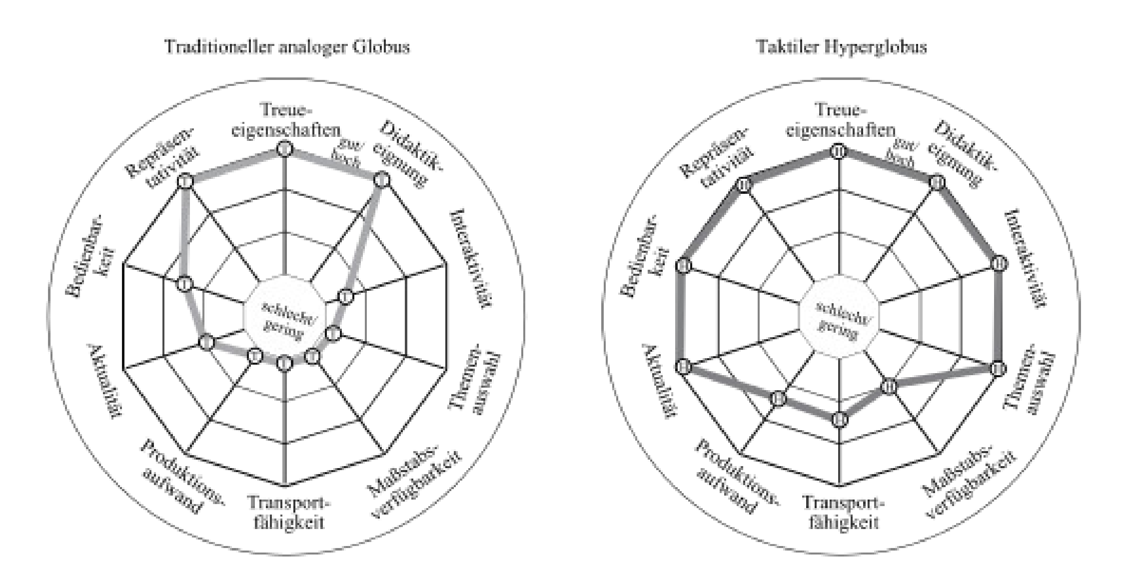


Abbildung 2: Globenprofile eines analogen Globus und eines taktilen Hyperglobus (Quelle: [RIE-00]:39 und [RIE-00]:131)

Da sich diese Arbeit speziell auf die taktilen Hypergloben bezieht, werden die virtuellen Hypergloben und die Hologloben im Folgenden nur kurz charakterisiert. Eine ausführliche Beschreibung findet sich bei [RIE-00].

2.2 VIRTUELLE HYPERGLOBEN

Ein virtueller Hyperglobus besitzt ausschließlich virtuelle Komponenten. Sowohl der Globenkörper, als auch das Abbild und der Raum, in dem sich der Globus befindet, sind nicht materiell bzw. real.

Virtuelle Hypergloben bestehen in der Regel aus einer Darstellung des Globus auf einem Bildschirm mit Hilfe eines Computers. Diese Technik ist nicht sehr aufwendig oder anspruchsvoll, so dass solche Globen schon seit geraumer Zeit existieren. Allerdings werden sie kaum noch zur Betrachtung der ganzen Erde (oder eines sonstigen Gestirns) verwendet, sondern dienen hauptsächlich als Interface für die großmaßstäbliche Betrachtung global verfügbarer und hochaufgelöster Datensätze (vgl. [RIE-08]). Deutlich wird dies an einem derzeit weit verbreiteten virtuellen Globus, der Software „Google Earth". In diesem Programm wird die Abbildung der gesamten Erde als Ausgangspunkt verwendet, von der der Benutzer auf einen Bereich seiner Wahl zoomt. Diese Betrachtung eines Ausschnitts entspricht dadurch nicht mehr der Definition eines Globus, sondern stellt eine andere kartographische Ausdrucksform dar. Trotzdem werden solche Produkte im allgemeinen Sprachgebrauch häufig als virtuelle Globen bezeichnet. HRUBY erläutert diese Verwechslung bzw. fehlerhafte Bezeichnung in [HRU-09].

2.3 HOLOGLOBEN

Betrachtet man die oben erwähnten Globenprofile, stellt sich heraus, dass ein Hologlobus - als einziger Globentyp - in allen Kategorien die bestmögliche Bewertung erzielt. Damit lässt er sich als idealer Globus bezeichnen (siehe [RIE-00]). Jedoch ist es nach dem heutigen Stand der Technik nicht möglich ein holographisches (sphärisches) Display zu verwirklichen und der Hologlobus bleibt somit bis auf weiteres ein Gedankenexperiment.

2.4 TAKTILE HYPERGLOBEN

Die taktilen Hypergloben, die eine Teilmenge der digitalen Globen darstellen, lassen sich als direkte Nachfolger des analogen Globus bezeichnen. Betrachtet man die Einteilung der digitalen Globen, wie sie in Abbildung 1 erfolgt, weisen sie die größte Übereinstimmung auf und unterscheiden sich lediglich in der Art der Abbildung des Inhalts auf dem Globenkörper.

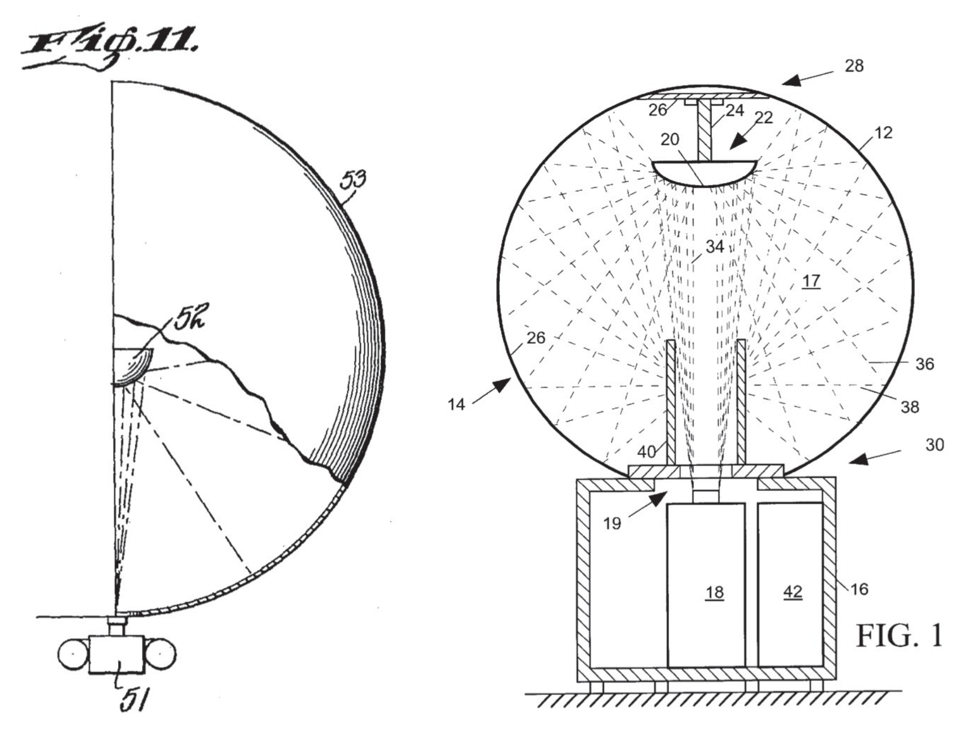


Abbildung 3: Gegenüberstellung von Patentskizzen: links US-Patent 2299682 aus dem Jahre 1942, rechts internationales Patent WO02065207A1 von 2002 (Quelle: [RIE-11]:5)

Wie ein Patent aus dem Jahr 1942 belegt (siehe Abbildung 3), ist die Idee eines sphärischen Displays schon älter. Jedoch konnten solche Displays damals noch nicht verwirklicht werden. Analog zur Situation des Hologlobus heute, bedurfte es erst einiger technischer Innovationen. Im Falle des patentierten Displays waren dies die modernen Videoprojektoren (Beamer).

Die Möglichkeiten taktile Hypergloben zu gestalten erschöpfen sich aber nicht allein in der oben angeführten Methode der Projektion eines Bildes auf eine Kugel über den Umweg eines oder mehrerer Spiegel. Die derzeit erhältlichen taktilen Hypergloben können nach ihren zu Grunde liegenden Prinzipien in drei Gruppen eingeteilt werden (vgl. [RIE-08]):

- Direkte Projektion
- Innenprojektion

- Außenprojektion

Während bei der direkten Projektion der Globenkörper sowohl der Ort der Abbildung als auch der Ort der Entstehung des Bildes ist, werden bei den beiden anderen Methoden extern erzeugte Bilder auf einen Globenkörper projiziert. Dies erfolgt durch die Verwendung unterschiedlicher optischer Systeme wie Linsen, Prismen und Spiegel. Auf Grund dieser verschiedenen Methoden weisen diese Typen von taktilen Hypergloben unterschiedliche Eigenschaften auf, die je nach Einsatzzweck des Globus einen Vorteil oder Nachteil darstellen können. Auf einige dieser Eigenschaften wird in den folgenden Kapiteln bei der Beschreibung der einzelnen Typen taktiler Hypergloben eingegangen. Eine vollständige Übersicht liefert RIEDL in [RIE-08].

2.4.1 Außenprojektion

Bei diesem Typus eines taktilen Hyperglobus besteht der Globenkörper aus einer hohlen, kugelförmigen Grundform aus transparentem Kunststoff. Auf der Außenseite ist eine Beschichtung ähnlich der einer Leinwand aufgebracht, um eine kontrastreiche Wiedergabe des Bildes auf dem ansonsten durchsichtigen Material zu ermöglichen. Der Globenkörper ruht entweder auf einem Sockel, kann aber auch hängend montiert werden. Die Bilder zur Darstellung der Inhalte werden von mehreren Videoprojektoren erzeugt, die in einer regelmäßigen Anordnung außen um den Globenkörper verteilt sind. Typischerweise kommen vier, fünf oder sechs Beamer zum Einsatz. Vier Projektoren sind in gleichen Abständen auf der Äquatorebene angeordnet und zum Zentrum des Globenkörpers hin ausgerichtet. Weitere Projektoren befinden sich in der verlängerten Achse zwischen Nord- und Südpol des Globenkörpers und sind ebenfalls auf dessen Mittelpunkt ausgerichtet.

Durch die große Anzahl der verwendeten Projektoren lässt sich zum einen eine sehr große Auflösung, also die Anzahl der einzelnen Pixel die auf dem Globenkörper abgebildet werden, erzielen und zum anderen auch hohe Werte für die Pixeldichte (entspricht der Anzahl der Pixel pro Fläche) erreichen. Daher wird diese Bauart überwiegend für große Kugeldurchmesser ab zwei Meter eingesetzt.

Allerdings ergeben sich auch einige Nachteile. Um eine gleichmäßige und lückenlose Projektion der Bilder über einen möglichst großen Bereich der Oberfläche des Globenkörpers zu erreichen ist es notwendig, dass sich die Projektionsbereiche der einzelnen Projektoren an ih-

ren Rändern überschneiden. Diese Überlappungsbereiche stellen besondere Anforderungen an die Kalibrierung der Hard- und Software. Um Fehler in der Darstellung zu vermeiden, müssen diese Bereiche in der Software zur Bilderzeugung besonders berücksichtigt werden. Ebenso müssen die Beamer im Millimeterbereich genau positioniert und ausgerichtet werden. Da der Globus und die dazugehörigen Komponenten auf Umwelteinflüsse, wie Luftfeuchtigkeit, Temperaturschwankungen oder Erschütterungen, reagieren sind regelmäßige Kalibrierungen notwendig. Durch eine qualitativ hochwertige Befestigung der Komponenten lassen sich die Intervalle in denen eine Nachtjustierung erfolgen muss verlängern.

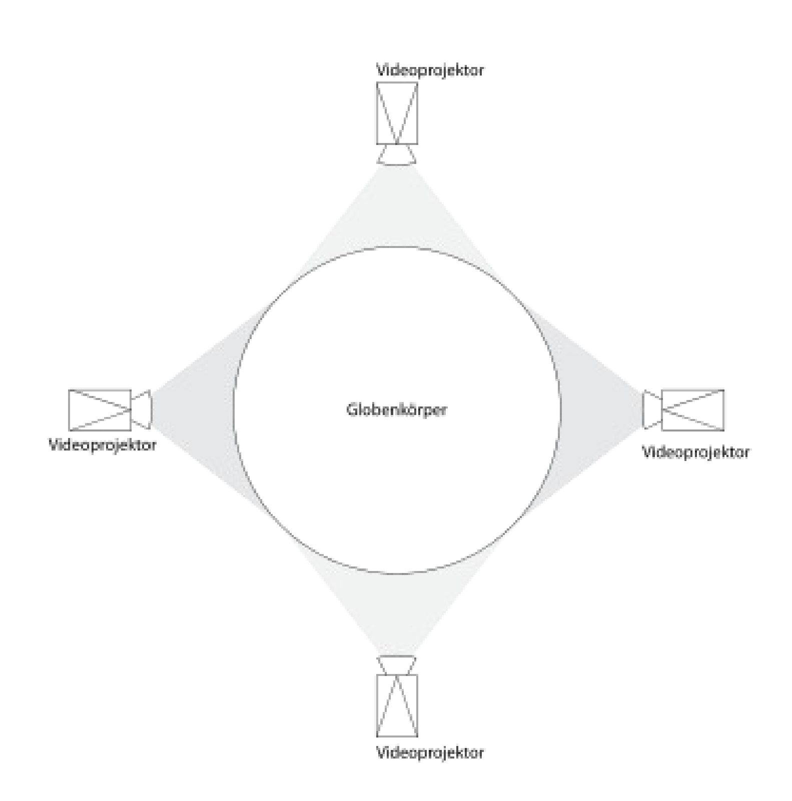


Abbildung 4: Schematischer Aufbau eines THG mit Außenprojektion, Draufsicht (Quelle: eigene Darstellung)

Durch den Einsatz von sechs Projektoren ist es theoretisch möglich die gesamte Globenoberfläche zur Darstellung von Inhalten zu benutzen. In der Praxis entstehen aber durch die Befestigung des Globenkörpers an einem Sockel oder durch andere Elemente, die der Verankerung dienen, Bereiche am Globus, von denen aus keine direkte Sichtverbindung zu einem der Projektoren existiert. Diese bauartbedingten Abschattungen werden „blind Spots" genannt. Ein anderer Nachteil der taktilen Hypergloben auf Basis der Außenprojektion ist, dass der Benutzer nicht direkt vor dem von ihm betrachteten Bereich stehen kann, da sonst sein Körper

ebenfalls eine Abschattung des Beamerstrahls verursachen würde. Man versucht diesen Effekt zu vermindern, indem der Globus so an einer Decke hängend montiert wird, dass sich der Äquator in etwa in Augenhöhe eines Benutzers befindet. Außerdem steigt mit dem Durchmesser des Globus auch die optimale Betrachtungsdistanz, so dass die Benutzer in der Regel einen Abstand einhalten, bei dem solche Abschattungen nicht hervorgerufen werden.

2.4.2 Innenprojektion

Aus den oben abgebildeten Patentskizzen (siehe Abbildung 3) geht hervor, dass das Modell eines taktilen Hyperglobus auf Basis einer Innenprojektion schon seit einiger Zeit existiert. Annähernd seit dem Jahr 2000 werden solche Globen in Kleinserie hergestellt und sind seitdem erhältlich. Das zu Grunde liegende Prinzip, die Projektion von Bildern auf die Innenseite einer hohlen Kugel lässt sich durch zwei verschiedene Methoden verwirklichen. Diese werden im folgenden Abschnitt beschrieben.

2.4.2.1 Projektion mittels Spiegel

Bei diesem Globentypus ist der Globenkörper ähnlich aufgebaut wie jene der Außenprojektions-Globen. Er besteht aus einer Hohlkugel aus transparentem Kunststoff, auf deren Außenseite eine Beschichtung aufgebracht ist, welche die Eigenschaften einer Leinwand besitzt. Auch die Montage des Globenkörpers erfolgt analog zum Außenprojektions-Globus. Entweder wird die Kugel auf einem Sockel ruhend oder abgehängt, wie z.B. von einer Decke, befestigt. An einem der Pole befindet sich eine Eintrittsöffnung durch die der Beamerstrahl ins Innere der Kugel eintritt. Am gegenüberliegenden Pol befindet sich ein halb-kugelförmiger, konvexer Spiegel, der im Folgenden als Disperser bezeichnet wird. Der Strahlengang des Lichts verläuft demnach vom Projektor, der sich außerhalb des Globenkörpers befindet, durch die Eintrittsöffnung hin zum Disperser. Von dort wird das Licht auf die Innenseite der Kugel reflektiert. Das Bild erscheint schließlich auf der außen angebrachten Beschichtung.

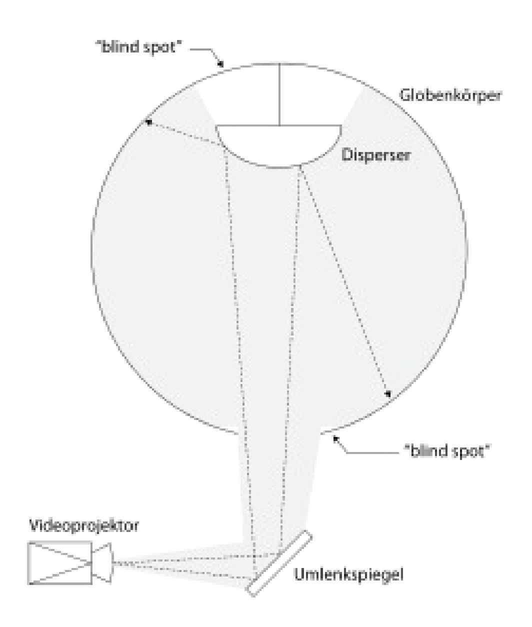


Abbildung 5: Schematischer Aufbau eines THG mit Innenprojektion, Querschnitt (Quelle: eigene Darstellung)

Auf Grund der Eintrittsöffnung und des Dispersers sind „blind Spots“ an beiden Polen bei dieser Art von Globen unvermeidbar. Zu einer Abschattung durch den Benutzer, wie bei der Außenprojektion, kommt es hierbei aber nicht. Allerdings tritt durch die Reflexion des Lichts am Disperser ein Verzerrungseffekt auf. In dem vom Beamer ausgestrahlten Bild weisen die einzelnen Pixel eine (nahezu) rechteckige Form auf. Durch die konvexe Krümmung des Dispersers wird die Geometrie der abgebildeten Pixel jedoch verändert und sie erscheinen auf der Globenoberfläche nicht mehr als Rechtecke. RIEDL erläutert und demonstriert diesen Effekt in [RIE-08] (siehe Abbildungen 6 und 7). Durch die Verwendung von Projektoren mit höheren Auflösungen bzw. den Einsatz von zwei Projektoren lässt sich das Ausmaß der Verzerrung verringern.

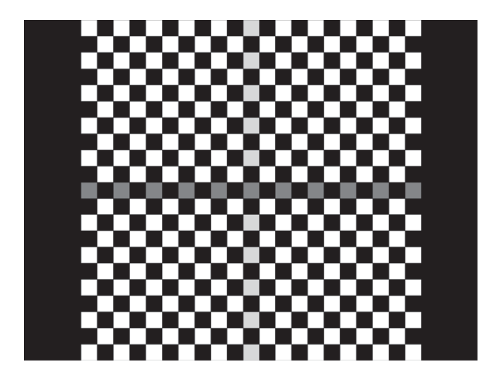


Abbildung 6: Testgrid für die projektionsbedingte (Pixel-)Verzerrung (Quelle: [RIE-08]:12)

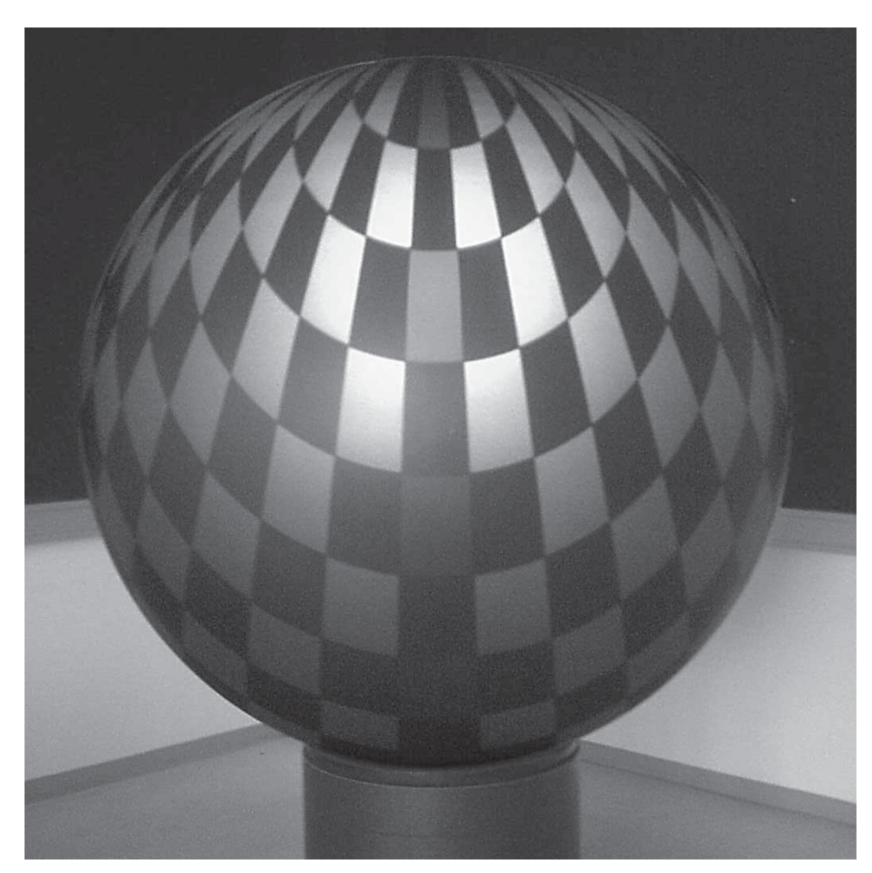


Abbildung 7: Projektionsbedingte (Pixel-)Verzerrung einer spiegelbasierten Innenprojektion (Quelle: [RIE-08]:12)

2.4.2.2 Projektion mittels Fischaugenobjektiv

Eine weitere Methode der Innenprojektion ist die Verwendung eines sogenannten Fischaugenobjektivs. Diese speziellen Objektive werden am Beamer angebracht und ersetzen die Standardoptik des Geräts. Durch ihre besondere Konstruktion weisen sie einen Abstrahlwinkel von ca. 180° auf. Der Globenkörper besitzt, wie bei der spiegelbasierten Innenprojektion, eine Eintrittsöffnung an einem der Pole. Dort wird der Projektor platziert und auf Grund der Abstrahlcharakteristik des Objektivs kann annähernd die ganze Kugel ausgeleuchtet werden. Diese Globen werden ebenfalls in verschiedenen Größen hergestellt und erreichen meist Durchmesser von einem bis fünf Metern.

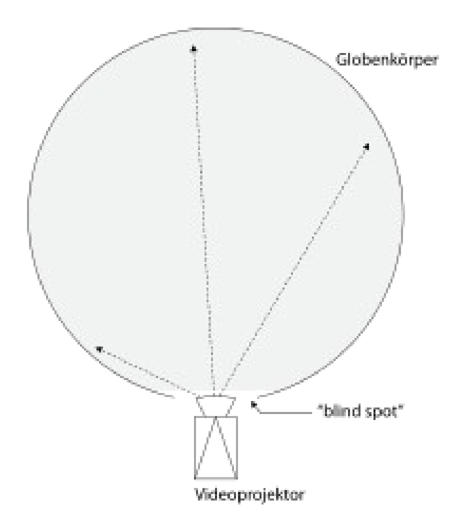


Abbildung 8: Schematischer Aufbau eines THG mit Fischaugenobjektiv, Querschnitt (Quelle: eigene Darstellung)

Einige dieser Globen können mit einem aufblasbaren Globenkörper ausgerüstet werden, wodurch die Transportfähigkeit äußerst positiv beeinflusst wird. Die Verwendung von mehreren Projektoren ist allerdings nicht möglich, wodurch dem Auflösungsvermögen sowie der Helligkeit des Globus Grenzen gesetzt sind.

Außerdem weisen diese denselben Verzerrungseffekt auf wie die oben beschriebenen spiegelbasierten Systeme.

2.4.3 Direkte Projektion

Bei allen bisher behandelten Arten von Hypergloben wird ein Bild von einem Projektor auf eine Kugel projiziert. Um einen möglichst großen Teil der Globenoberfläche beleuchten zu können, muss auf mehrere Projektoren oder optische Hilfsmittel zurückgegriffen werden.

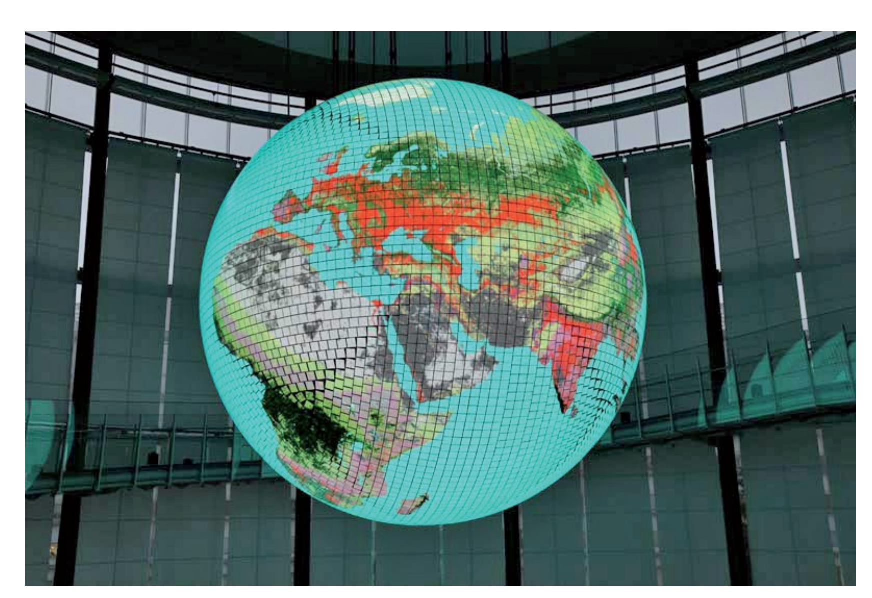


Abbildung 9: LED-Globus GeoCosmos II (6 m Durchmesser,10 Mpix Auflösung, Tokyo, Japan) (Quelle: [RIE-11]:6)

Durch diese zusätzlichen Komponenten erhöht sich der Aufwand für die Herstellung und Wartung. Eine „einfachere“ Methode ist die direkte Projektion. Einzelne Pixel werden nicht durch Projektion auf der Globenoberfläche abgebildet, sondern die aktiven Leuchtelemente befinden sich auf ihr. Dieser Globus stellt den idealen taktilen Hyperglobus dar, weil er völlig frei von „blind spots“ und Verzerrungen ist und durch den Verzicht auf den Einsatz von optischen Bauteilen keine Kalibrierung notwendig ist. Diese Methode lässt sich mit dem heutigen Stand der Technik aber nur schwer verwirklichen. Es existieren zwar einige Einzelmodelle, die aber in ihrer Konstruktion sehr aufwendig und teuer sind.

Bei dem in Abbildung 9 dargestellten Globus wurde eine Vielzahl von LED-Panelen verwendet, die zusammen ein annähernd kugelförmiges Konstrukt bilden. Eine andere Möglichkeit ist die Verwendung von Glasfaserleitern. Eine große Anzahl solcher Leiter wird so zusammengefasst, dass eines ihrer Enden jeweils einen Punkt der Globusoberfläche darstellt. Das jeweils andere Ende wird mit einer Lichtquelle verbunden, wodurch das Globenbild erzeugt wird. Durch den Fortschritt im Bereich der LED-Technik und vor allem bei den organischen Leuchtdioden sollten in absehbarer Zeit flexible LED-Module erhältlich sein, mit denen sich

ein kugelförmiger Grundkörper bedecken lässt. RIEDL zeigt sich in [RIE-11] davon überzeugt, dass diese Globen in Zukunft den größten Verbreitungsgrad erreichen werden.

3. Touchscreen-Systeme

Nach der Einführung in die verschiedenen Typen von Hypergloben im vorherigen Kapitel, soll nun an dieser Stelle ein Überblick über die Funktionsprinzipien der Touchscreen-Technologien folgen. Die Anzahl der Touchscreen-Systeme, die im Alltag anzutreffen sind, wächst stetig. Waren sie bisher hauptsächlich in Anwendungen wie Geld- und Fahrscheinautomaten, Museen, Supermarktkassen oder ähnlichen Szenarien verbaut, sind sie seit dem Aufkommen von Smartphones und Tablett-Computern heute nahezu in jedem Haushalt anzutreffen. Das hat zur Folge, dass ein überwiegender Teil der potentiellen Globennutzer bereits mit der Technik vertraut und den Umgang mit ihr gewohnt ist. Zu den Gründen für die weite Verbreitung von berührungssensitiven Displays zählt unter anderem, dass sie in vielen Fällen den klassischen Eingabemethoden (Kombination aus Tastatur und Maus) überlegen sind. Obwohl sie in der Anschaffung teurer sind, ist der Wartungsaufwand auf Grund der kompakten und geschlossenen Bauweise im praktischen Einsatz geringer. Die Vorteile eines Touchscreens hinsichtlich der Benutzerfreundlichkeit erschließen sich gut am Beispiel des Fahrkartenautomaten. Diese Systeme müssen auch für Menschen mit eingeschränkter Motorik oder einer Behinderung leicht zu bedienen sein und in der Regel gelingt es solchen Personen leichter einen Punkt auf einem Bildschirm zu berühren, als dieselbe Aktion mit einer Maus durchzuführen (vgl. [SNE-06]). Bei kleinformatigen Geräten, wie den oben erwähnten Smartphones, ist eine Bedienung über eine Tastatur auch für nicht eingeschränkte Anwender kaum noch sinnvoll umzusetzen. Bessere Bedienbarkeit erzielt man hier mit einem Touchscreen und einem dafür angepassten User-Interface.

In der Art der Steuerung unterscheiden sich Touchscreen-Systeme kaum voneinander. Sehr wohl jedoch hinsichtlich ihrer zu Grunde liegenden technischen Funktionsprinzipien. Um diese sinnvoll zu kategorisieren, ist es notwendig die Definition und die unabdingbaren Komponenten eines Touchscreen-Systems zu betrachten.

Touchscreens werden auch als kombinierte Ein- und Ausgabegeräte bezeichnet, da sie einen bidirektionalen Informationsfluss verarbeiten. Die Ausgabekomponente ist dabei die Bilderzeugung, die Eingabekomponente die Erkennung von Berührungen.

> *Even though forms of gestural devices can vary wildly—from massive touch screens to invisible overlays onto environments—every device* [...] *has at least three general parts: a sensor, a comparator, and an actuator. These three parts can be a single physical component, or, more typically, multiple components of*

> *any gestural system, such as a motion detector (a sensor), a computer (the comparator), and a motor (the actuator).*
> [SAF-08]:12

SAFFER listet die drei Komponenten auf, über die jedes durch Gesten gesteuerte System mindestens verfügt (vgl. [SAF-08]):

- Sensor (sensor)
- Auswerteeinheit (comparator)
- Reaktionseinheit (actuator)

Im Falle eines Touchscreen-Systems kommen zusätzlich hinzu:

- Touchoberfläche
- Bildfläche
- Bilderzeuger
- Emitter

Durch diese funktionalen Komponenten lässt sich der prinzipielle Aufbau eines Touchscreen-Systems und die Vorgänge bei einer Berührung darstellen. Die Bezeichnung „funktional" wurde gewählt, da bei einigen Touchscreen-Modellen mehrere Komponenten in einem einzelnen Bauelement zusammengefasst werden. Abbildung 10 zeigt einen solchen schematischen Aufbau.

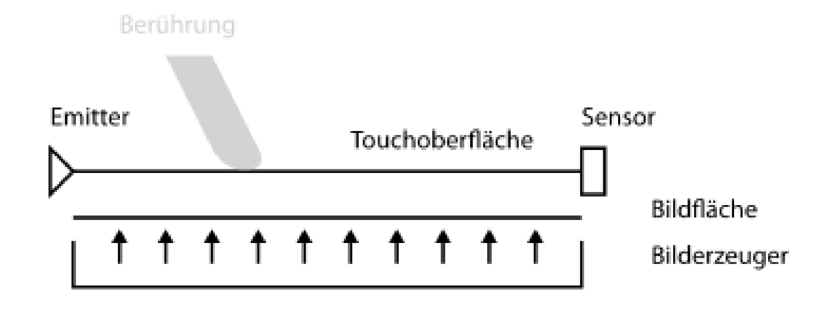


Abbildung 10: Schematischer Aufbau eines Touchscreens (Quelle: eigene Darstellung)

Aus der Sicht eines Benutzers ist die oberste Komponente die transparente Touchoberfläche. Sie ist der Ort, an dem eine Berührung stattfindet. Mit der Touchoberfläche verbunden sind ein oder mehrere Emitter und Sensoren. Darunter befinden sich die Bildfläche und der Bilderzeuger. Als Bilderzeuger werden meist herkömmliche Monitore (TFT, LCD, Röhrenmonitore) verwendet. Die Bildfläche und der Bilderzeuger bilden dann einen Verbund. Andere Systeme werden mit Videoprojektoren betrieben und die Bildfläche ist physisch vom Bilderzeuger getrennt und fungiert bei einigen Typen gleichzeitig als Touchoberfläche. Die von SAFFER erwähnten Reaktions- und Auswerteeinheiten werden im Bereich der hier besprochenen Touchscreen-Systeme durch Computerhardware und entsprechender Software verkörpert. Durch eine Berührung der Touchoberfläche werden die vom Emitter ausgesendeten Signale beeinflusst und diese Veränderung durch den Sensor registriert. Diese Sensorwerte werden innerhalb der Steuerungssoftware (Reaktions- und Auswerteeinheit) ausgewertet und die Koordinaten der Berührung berechnet. Ist ein System in der Lage mehrere simultane Berührungen unabhängig voneinander zu registrieren, spricht man von einem System mit Multitouch-Fähigkeit.

Anhand der Emitter und Sensoren – bzw. die Art der Signale, die von den Sensoren registriert werden - lassen sich die verschiedenen Touchscreen-Technologien in drei Gruppen unterscheiden:

- elektrische Sensorik
- akustische Sensorik
- optische Sensorik

Im Folgenden werden diese drei Gruppen ausführlicher behandelt, die gängigsten Technologien vorgestellt, ihre Vor- und Nachteile beschrieben und ihre Eignung für den Einsatz an einem taktilen Hyperglobus beleuchtet. Die Details zu den Grundprinzipien und Eigenschaften der verschiedenen Systeme finden sich in den Arbeiten von [MÖS-11], [SNE-06] und [ODE-10].

3.1 ELEKTRISCHE SENSORIK

In dieser Gruppe wird eine Berührung durch die Messung von Effekten bestimmt, die durch einen elektrischen Stromfluss hervorgerufen werden.

3.1.1 RESISTIVE SYSTEME

Resistive Touchscreens haben sich auf Grund ihrer relativen Einfachheit und anderer Vorteile als erste Touch-Technologie auf dem Markt durchgesetzt. Sie gelten daher als ausgereift und zuverlässig. Das Kernstück dieser Methode bilden drei übereinanderliegende, transparente Schichten, von denen die oberste (erste Schicht aus der Sicht des Benutzers) und die unterste Schicht elektrisch leitend sind (siehe Abbildung 11). Die mittlere Schicht besteht in der Regel aus Silikon-Abstandhaltern und sorgt dafür, dass die beiden leitenden Schichten in Normalzustand voneinander getrennt sind. Diese drei Lagen bilden zusammen die Touchoberfläche. An den leitenden Schichten sind zusätzlich zwei horizontale bzw. vertikale Elektroden angebracht, über die eine anliegende Spannung und Stromstärke gemessen werden kann. Sobald der Touchscreen berührt wird, werden durch den dabei entstehenden Druck die Abstandhalter zusammengepresst und es kommt zum Kontakt zwischen den leitenden Schichten. Um die Position der Berührung zu ermitteln, wird abwechselnd zwischen zwei der vier Messelektroden eine definierte Spannung angelegt und der Stromfluss gemessen. Aus dem Verhältnis von Stromstärke zur Spannung lässt sich der elektrische Widerstand berechnen. Da sich dieser Widerstand direkt proportional zur Länge des vom Strom durchflossenen Leiters verhält, kann aus den errechneten Werten auf die Koordinaten des berührten Punkts geschlossen werden. Diese Methode stellt die einfachste und älteste Form der resisitiven Touchscreens dar. Mittlerweile existieren auch noch weitere Umsetzungen, die dem oben beschriebenen Prinzip jedoch ähneln, und deren Aufzählung und Erklärung dem Rahmen dieser Arbeit nicht angemessen ist (vgl. [ODE-10]).

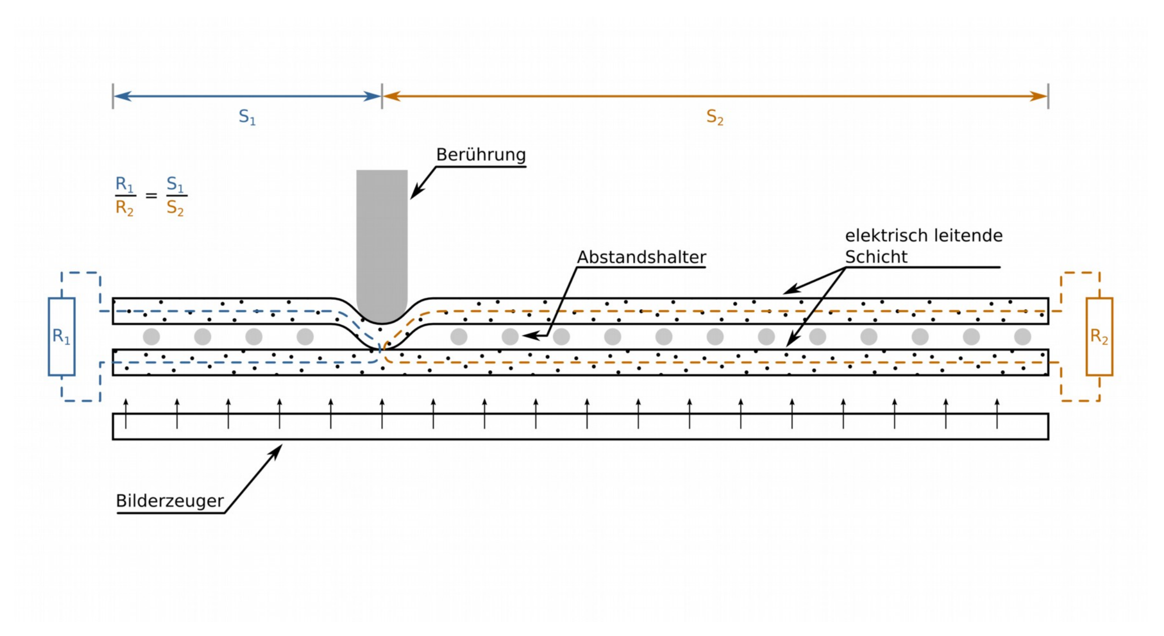


Abbildung 11: vereinfachtes Funktionsprinzip eines resistiven Touchscreens (Quelle: eigene Darstellung)

Zu den Vorteilen dieser Technik zählt, dass sie unempfindlich gegenüber äußeren Einflüssen wie z.B. Ablagerung von Staub oder Wasser auf der Oberfläche sind. Zudem können Bildschirme durch ein vorgelagertes Touch-Modul leicht nachgerüstet werden. Allerdings sind solche Touchscreen-Systeme auf Grund ihrer Bauart nicht in der Lage mehrere simultane Berührungen unabhängig voneinander zu erkennen und verfügen daher über keinerlei Multitouch-Fähigkeit (ebenfalls [ODE-10]). Außerdem können die notwendigen Schichten nicht vollständig durchsichtig konstruiert werden. Sie absorbieren einen Teil des vom Bildschirm ausgehenden Lichts, wodurch eine Trübung des dargestellten Bildes erfolgt.

Eingesetzt werden sie vor allem in Umgebungen mit besonders anspruchsvollen Bedingungen wie z.B. öffentlich zugängliche Automaten, Reinräumen und Operationssälen. Betrachtet man die Vor- und Nachteile dieser Technik, kommt man zu dem Schluss, dass sie für den Einsatz an einem taktilen Hyperglobus nicht geeignet ist. Zum einem ist der Verlust an Leuchtkraft durch die Trübung und die fehlende Multitouch-Fähigkeit problematisch und zum anderen sind diese Systeme auf die Anwendung bei flachen Bildschirmen ausgelegt. Die Anpassung an die Kugelform des Globus würde einen wirtschaftlich nicht zu rechtfertigenden Aufwand erfordern.

3.1.2 KAPAZITATIVE SYSTEME

Kapazitative Touchscreens benötigen im Vergleich zu den resistiven Modellen nur eine spezielle, transparente Schicht, die ebenfalls direkt vor dem Bildschirm angebracht ist und die Touchoberfläche darstellt. Diese Schicht verhält sich wie ein Kondensator. Sie kann also elektrische Ladungen aufnehmen, über einen begrenzten Zeitraum hinweg speichern und wieder abgeben. An den vier Ecken dieser Schicht befinden sich Elektroden, über die entweder Stromstärken oder die Ausrichtung von elektromagnetischen Feldern detektiert werden können. Im Betrieb wird eine kleine Spannung an den Elektroden angelegt und die kapazitative Schicht wie ein Kondensator aufgeladen. Bei einer Berührung mit einem leitfähigen Material geht ein kleiner Teil der Ladung auf das berührende Objekt über. Da an den Elektroden eine Spannung anliegt, fließt Strom in Richtung des Kondensators, um den Ladungsverlust auszugleichen. Die an die Elektroden angeschlossenen Sensoren, registrieren diesen Stromfluss bzw. die Richtung des dadurch entstehenden elektromagnetischen Felds. Aus diesen Sensorwerten lässt sich dann der Punkt der Berührung bestimmen.

Touchscreens dieser Bauart sind Multi-Touch-fähig und absorbieren im Gegensatz zu resistiven Touchscreens nahezu kein Licht. Sie erscheinen also vollständig transparent. Ein bedeutender Nachteil dieser Technik ist allerdings, dass nur Berührungen durch elektrisch leitende Gegenstände erkannt werden. Daher lassen sich zum Beispiel Smartphones und Tablett-Computer, den momentanen Haupteinsatzgebiet dieser Technologie, nicht bzw. nur mit speziellen Handschuhen bedienen (vgl. [SNE-06]).

Prinzipiell ließe sich ein taktiler Hyperglobus mit einer kapazitativen Touchscreen-Oberfläche ausrüsten. Allerdings wäre, analog zu den resistiven Touchscreens, die Herstellung eines solchen Modells wirtschaftlich wohl nicht rentabel, da die Aufbringung auf eine Kugel einen nicht alltäglichen Sonderfall darstellt.

3.2 AKUSTISCHE SENSORIK

Ähnlich der Vorgehensweise bei den kapazitativen Touchscreens werden bei den Modellen, die mit der sogenannten SAW-Technik („Sourface acoustic wave“) ausgestattet sind, Berührungen durch die Absorption von Energie erkannt. Eine Schicht aus Glas dient als Touchoberfläche und wird mit Ultraschallwellen in Schwingung versetzt. Bei einer Berührung dieser Schicht wird ein Teil der Wellenenergie auf den berührenden Körper übertragen. Dadurch ergeben sich Veränderung im Ausbreitungsmuster der Schallwellen im Glas. Diese Variationen werden von Sensoren als Änderung des Schalldrucks gemessen. Durch die vergleichende Auswertung der Messwerte von unterschiedlich positionierten Sensoren (üblicherweise vier Sensoren in den Ecken des Touchscreens) lässt sich der Punkt der Berührung errechnen.

Da die Touchoberfläche bei dieser Technik aus einer einfachen Glasscheibe besteht, die keine zusätzliche Beschichtung benötigt, wird das Licht des Monitors und damit die Bildqualität nicht beeinflusst. Allerdings reagiert diese Methode empfindlich auf mechanische Beschädigungen der Glasscheibe, wie etwa Kratzer und Ablagerungen, da diese das Ausbreitungsverhalten der Schallwellen störend verändern. Die Erkennung von mehreren gleichzeitig stattfindenden Berührungen ist möglich und somit die Multitouch-Fähigkeit gegeben (vgl. [MÖS-11]).

Gegen den Einsatz in Verbindung mit einem taktilen Hyperglobus spricht wiederum die zu erwartende sehr aufwendige Umsetzung auf einer Kugel.

3.3 OPTISCHE SENSORIK

Bei diesen Systemen kommen Infrarot-Lichtquellen als Emitter zum Einsatz. Eine Berührung der Touchoberfläche hat je nach System eine Zu- oder Abnahme der an den Sensoren registrierten Strahlungsintensität zur Folge. Als Sensoren werden einfache IR-Detektoren oder auch im Infrarot-Spektrum empfindliche Videokameras verwendet, wobei unter einem IR-Detektor eine Kamera mit einer Auflösung von lediglich einem Pixel verstanden werden kann. Im Folgenden werden drei verschiedene Systeme beschrieben. Das sogenannte Leuchtgitterverfahren, welches mit herkömmlichen Monitoren kombiniert wird, sowie zwei weitere Methoden, die mit einem Rückprojektionsverfahren verwendet werden.

Vor allem die beiden letztgenannten Systeme lassen sich leicht skalieren und sind somit für

großformatige Multitouch-Anwendungen geeignet. Häufig werden sie in nicht kommerziellen Projekten und Hobbyanwendungen eingesetzt, da sie weder teure noch aufwendige Komponenten erfordern und sich gut zum Aufbau von Prototypen eignen.

3.3.1 LEUCHTGITTER SYSTEME

Diese Methode arbeitet nach dem Prinzip einer Lichtschranke. Vor einem Bilderzeuger werden auf zwei nicht gegenüberliegenden Seiten Infrarot-Emitter, in der Regel Leuchtdioden, die Licht im Infrarotspektrum aussenden, angebracht. Auf den jeweils gegenüberliegenden Seiten befinden sich IR-Detektoren. Durch eine Berührung der Touchoberfläche (in diesem Fall identisch mit der Bildfläche) erfolgt eine Abschattung der emittierten Strahlung. Diese wird als Verringerung der Strahlungsintensität an den Sensoren registriert. Da die Position der Sensoren bekannt ist, kann die Position der Berührung bestimmt werden. Diese Methode ist nur bedingt multitouchfähig, da maximal zwei simultane Berührungen immer zweifelsfrei zugeordnet werden können. Eine Besonderheit dieser System ist, dass keine zweidimensionale Touchoberfläche vorhanden ist, sondern ein dredimensionaler „Touchraum“ vor dem Bilderzeuger entsteht. Zum Erkennen einer Berührung ist also kein direkter Kontakt mit einer Touchober- bzw. Bildfläche nötig (vgl. [MÖS-11]).

Eine Verwendung dieser Methode mit einer gekrümmten Bildfläche ist nur sehr eingeschränkt und ab einem bestimmten Krümmungsgrad nicht mehr möglich. Daher ist sie auch nicht für den Einsatz an einem taktilen Hyperglobus geeignet.

3.3.2 FTIR (FRUSTRATED TOTAL INTERNAL REFLECTION)

Diese Systeme nutzen den Effekt der Totalreflexion, um eine Änderung der Strahlungsintensität in Folge einer Berührung zu erreichen. Die Totalreflexion beschreibt das Verhalten von elektromagnetischer Strahlung (in diesem Fall Infrarotlicht) beim Übergang von einem transparentem Material in ein anderes transparentes Material mit einem geringeren Brechungsindex. Ist der Winkel (siehe α, β und γ in Abbildung 12) zwischen der Grenzfläche des Materials und dem Lichtstrahl kleiner als ein von den beteiligten Materialien abhängiger Grenzwert wird das Licht fast vollständig reflektiert und verbleibt im optisch dichteren Material.

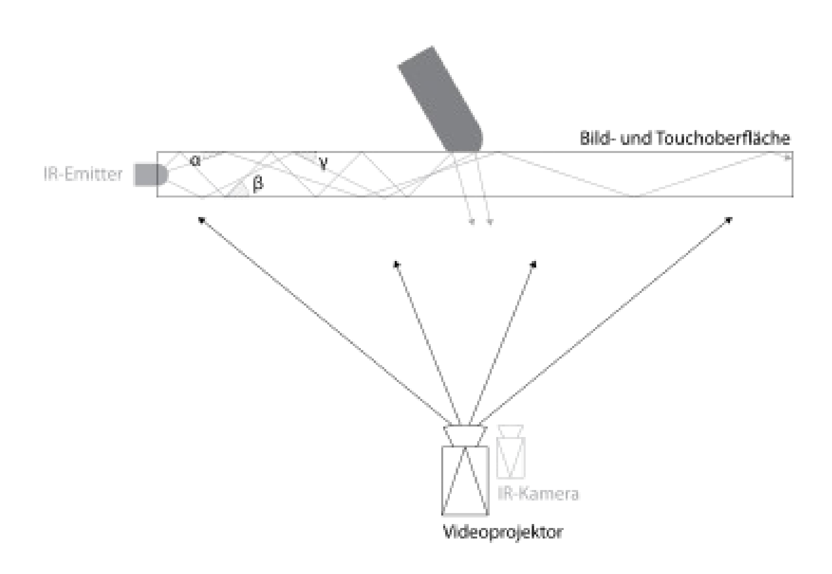


Abbildung 12: Schematischer Aufbau eines FTIR-Touchscreen-Systems (Quelle: eigene Darstellung)

Der schematische Aufbau eines solchen Touchscreen-Systems ist in Abbildung 12 dargestellt. Die Bild- und Touchfläche bilden eine Einheit. Meistens besteht diese aus Glas oder Acrylglas (Plexiglas), die ähnlich der im zweiten Kapitel beschriebenen Globenkörper auf der Außenseite mit einer diffusen Beschichtung versehen ist, um eine Darstellung des Bildes auf der Oberfläche zu erreichen. In diese Bildfläche wird seitlich Infrarotlicht eingebracht. Als Bilderzeuger wird ein Videoprojektor eingesetzt, der sich hinter der Bildfläche befindet (Rückprojektion). Ebenfalls hinter der Bildfläche ist eine Videokamera angebracht, die nur im Spektrum der verwendeten Infrarotstrahlung empfindlich ist.

Im berührungslosen Zustand verbleibt das Infrarotlicht auf Grund des oben beschriebenen Phänomens der Totalreflexion innerhalb der Bildfläche, da der Brechungsindex der Luft ge-

ringer ist als der der Bildfläche. Bei einer Berührung dieser Fläche durch ein Objekt mit höherer optischer Dichte wird an dieser Stelle die Totalreflexion unterbrochen (englisch: frustrated total internal reflexion) und das Infrarotlicht vom Objekt reflektiert. Diese reflektierte Strahlung wird von der als Sensor fungierenden Infrarotkamera aufgenommen. Die Bestimmung der Position der Berührung erfolgt anschließend durch eine Bilderkennungssoftware.

Im Gegensatz zur Leuchtgittermethode können bei diesem Verfahren nahezu beliebig viele simultane Berührungen unterschieden werden. Der einzige limitierende Faktor ist die Auflösung der Infrarotkamera. Ein Nachteil der Rückprojektionssysteme ist ihr Platzbedarf. Da hinter der Bildfläche die IR-Emitter, die Infrarotkamera und vor allem der Videoprojektor positioniert werden müssen, können solche Systeme nicht annähernd so flach gebaut werden wie andere Touchscreen-Systeme, die auf herkömmliche Flachbildmonitore als Bilderzeuger zurückgreifen. Für größere Anwendungen sind sie jedoch besser geeignet, da sich mit Projektoren größere Bildflächen leichter umsetzen lassen. Für die Verwendung mit taktilen Hypergloben ist diese Methode sogar sehr gut geeignet, da sich die wenigen benötigten Komponenten (IR-Emitter und Infrarotkamera), bei dem den Autor bekannten Modellen, mit wenig Aufwand integrieren lassen. Nur bei einer Nachrüstung eines solchen Systems können Probleme auftreten, da die Installation der IR-Leuchtdioden nicht immer ohne Eingriffe in die Struktur des Globus möglich ist (vgl. [HAL-10]).

3.3.3 DI (DIFFUSE ILLUMINATION)

Eine weitere Rückprojektionsmethode ist jene der diffusen Illumination. Aus der Abbildung 13 geht der schematische Aufbau eines solchen Systems hervor. Analog zum oben beschriebenen FTIR-System bilden die Bild- und Touchoberfläche eine Einheit. Auch die Anordnung des Videoprojektors und einer Infrarotkamera, die als Sensor dient, entspricht jener der FTIR-Systeme. Um eine Änderung der Strahlungsintensität durch eine Berührung zu generieren, wird bei dem Verfahren der diffusen Illumination Infrarotlicht hinter der Bildfläche emittiert, sodass es im normalen Zustand (ohne Berührung) die Bildfläche ohne Reflexion durchdringt. Sobald die Bildfläche von einem Objekt berührt wird, wird an dieser Stelle das Infrarotlicht reflektiert und von der Infrarotkamera registriert. Durch die Auswertung der Kamerabilder durch eine Bilderkennungssoftware wird die Position der Berührung bestimmt.

Diese Methode ist eng verwandt mit den FTIR-Systemen, wodurch sie in puncto Multitouch-Fähigkeit, Platzbedarf und Skalierbarkeit die gleichen Eigenschaften aufweisen. Eine Beson-

derheit der DI-Methode ist, dass, ähnlich wie bei der Leuchtgittermethode, Berührungen zu einem gewissen Teil schon erkannt werden bevor ein direkter Kontakt zur Bildfläche besteht (vgl. [HAL-10]).

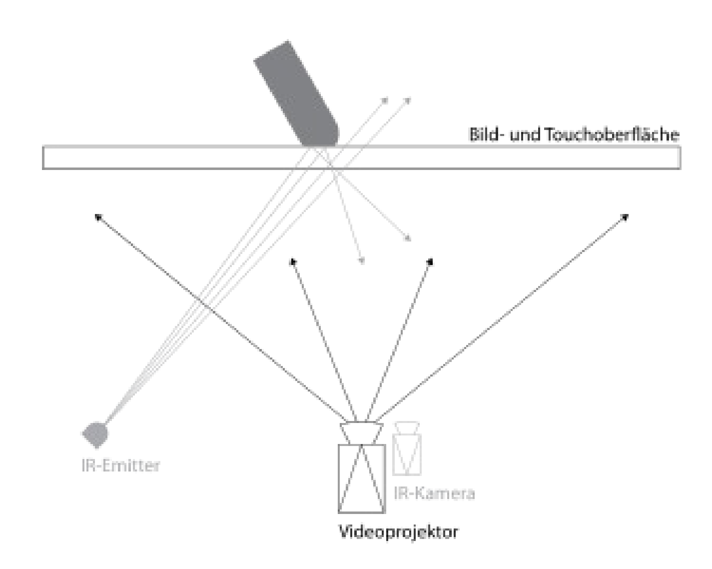


Abbildung 13: Schmatischer Aufbau eines DI-Touchscreen-Systems (Quelle: eigene Darstellung)

Von allen bisher besprochenen Touchscreen-Systemen erscheint dieses als am besten geeignet für den Einsatz mit einem taktilen Hyperglobus, da es sich auch in bereits bestehende Systeme mit minimalen Aufwand und ohne die Struktur des Globus zu verändern integrieren lässt.

4. Existierende berührungsempfindliche sphärische Displays

Die Idee eines berührungsempfindlichen sphärischen Displays ist weder neu, noch eine Eigenleistung des Autors. Eine Internetrecherche ergab, dass seit dem Jahr 2008 mindestens drei, zum Teil simultan entwickelte Projekte zu diesem Thema existieren. Sie weisen einen sehr unterschiedlichen Hintergrund auf. Er reicht von der Forschungsabteilung einer der weltweit größten und bekanntesten Softwarefirmen (Microsoft), über kommerzielle Anbieter bis hin zu universitären Forschungsprojekten. Durch diese Unterschiede, liegt die Vermutung nahe, dass die jeweiligen Projekte mit verschiedenen Zielsetzungen und voneinander abweichenden finanziellen und personellen Ressourcen umgesetzt wurden. Durch den Vergleich und die Analyse der drei Projekte und deren Herangehensweise an die Problemstellung erhofft sich der Autor Rückschlüsse für die eigene Umsetzung eines multitouchfähigen taktilen Hyperglobus ziehen zu können und eventuell auftretende Probleme, soweit möglich, schon im Vorfeld auszuschließen.

Die Verfügbarkeit von Informationen zu den einzelnen Projekten ist sehr unterschiedlich und zum Teil als dürftig zu bezeichnen. Trotzdem sollen sie im folgenden alle vorgestellt werden.

4.1 MICROSOFT SPHERE

Das unter dem Titel „Sphere“ veröffentlichte Projekt wurde von BENKO et al. innerhalb der Forschungsabteilung „Microsoft Research“ der Firma Microsoft durchgeführt. Die Ergebnisse wurden im Jahr 2008 in Form eines Papers veröffentlicht (siehe [BEN-08]). Eine kommerzielle Umsetzung in ein Produkt ist bisher nicht erfolgt. Der Anwendungsschwerpunkt liegt bei der

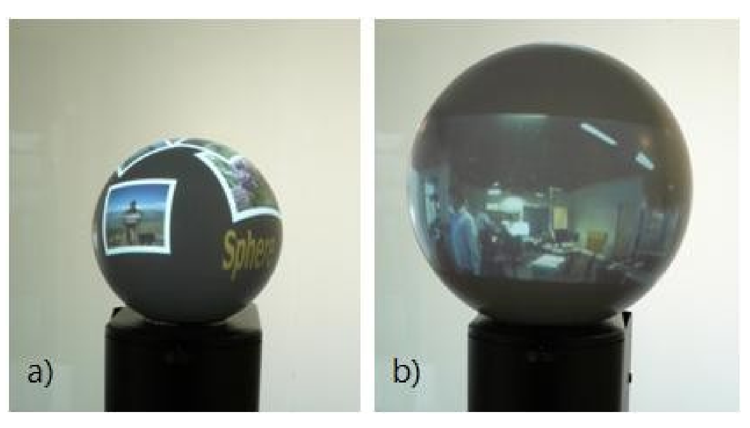


Abbildung 14: "Microsoft Sphere", Größenvergleich: links 16 Zoll, rechts 24 Zoll (Quelle: [BEN-08]:3)

„Sphere“ nicht auf der Darstellung von Geoinformationen, sondern sie ist als multitouch- und mehrbenutzerfähiges sphärisches Input- und Output-System konzipiert. Die speziellen Eigenschaften dieser Gattung von Anzeigegeräten verleiht ihnen, der Meinung von BENKO et al. zufolge, einige Vorteile gegenüber flachen Bildschirmen. Als Beispiel sei hier die Tatsache erwähnt, dass ein Benutzer ohne seinen Standpunkt zu verändern maximal eine Hälfte des Displays einsehen kann und somit zwei Personen gleichzeitig an einem Gerät agieren können, ohne sich gegenseitig zu beeinflussen. Außerdem wird dadurch eine Art „Pseudo-Privatsphäre“ geschaffen. Ein Benutzer weiß intuitiv welche Bereiche des Bildschirms von anderen eingesehen werden können und wie sich diese bei einer Positionsänderung verhalten.

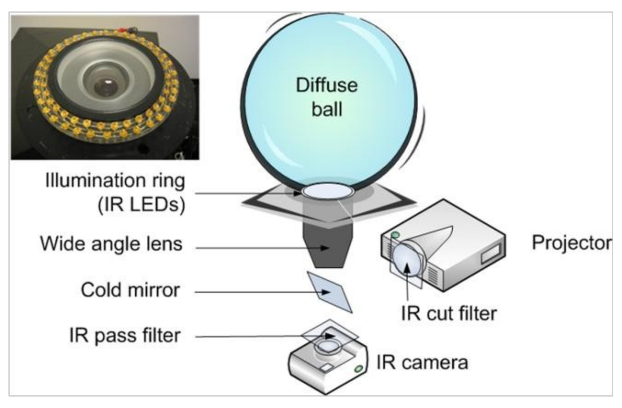


Abbildung 15: „Microsoft Sphere", schematischer Aufbau (Quelle: [BEN-08]:4)

Als Hardwareplattform wurden zwei taktile Hypergloben mit der Produktbezeichnung „Magic Planet" der Firma Global Imagination verwendet, die eine Innenprojektion durch ein Fischaugenobjektiv realisieren. Die Durchmesser der Globen sind mit 16 und 24 Zoll angegeben, dass einem Maß von 40 bzw. 61 cm entspricht. Die Auflösung der eingesetzten Projektoren beträgt 1400x1050 Pixel. Allerdings kann nicht der gesamte Projektionsbereich des Beamers auf dem Globus abgebildet werden, sondern nur eine Kreisfläche. Der Durchmesser dieses Kreises entspricht der kürzeren Seite des rechteckigen Projektionsbereiches. Effektiv werden daher am Globus nur ca. 866.000 Pixel dargestellt. Die Multitouch-Fähigkeit wurde durch ein nachträglich hinzugefügtes System, basierend auf der diffusen Illumination, erreicht. Aus Abbildung 15, die den schematischen Aufbau einer „Sphere" darstellt, geht die Besonderheit dieses Systems hervor: Das sichtbare Licht, das zur Bilderzeugung eingesetzt wird, und das zur Berührungserkennung verwendete Infrarotlicht teilen sich denselben optischen Pfad. Unterhalb des am imaginären Südpol angebrachten Fischaugenobjektivs werden die beiden Lichtspektren durch einen Kaltlichtspiegel, der sichtbares Licht reflektiert und infrarotes Licht passieren lässt, separiert und vom Videoprojektor zum Fischaugenobjektiv bzw. von diesem Objektiv hin zur Infrarotkamera geleitet. Die als Emitter fungierenden Infrarotleuchtdioden sind ringförmig an der Vorderseite des Weitwinkelobjektivs und damit innerhalb des Globenkörpers angebracht. BENKO et al. erläutern außerdem den Aufbau und die Funktionsweise der Software, die zur Bilderzeugung und zur Berührungserkennung eingesetzt wird. Um den logischen Aufbau der Arbeit an dieser Stelle nicht zu unterbrechen, werden diese Aspekte im fünften Kapitel behandelt.

Zum Testen der Multitouch-Funktionen entwickelten BENKO et al. einen speziellen Photo- und Videobrowser und implementierten folgende berührungsgesteuerte Funktionen:

- „Dragging“:
  Zum Verschieben eines Objektes (Photo oder Video) muss der Benutzer dieses Berühren und anschließend eine Bewegung hin zum gewünschten Zielort ausführen.

- „Rotation“:
  Wird ein Objekt an zwei Punkten berührt und eine Bewegung ausgeführt wird der Winkel zwischen der Ausgangs- und Endposition der Berührungen als Drehung auf das Objekt übertragen.

- „Scaling“:
  Analog zur „Rotation“ wird hier die Änderung der Distanz zwischen zwei Berührungspunkten als Größenänderung auf ein Objekt übertragen.

- „Flicking“:
  Diese Funktion stellt eine Erweiterung des „Dragging“ dar. Dabei folgt ein bewegtes Objekt nach Beendigung der Berührung weiter dem Bewegungspfad. Die Geschwindigkeit nimmt mit der Zeit ab (Trägheit), bis das Objekt zum Stillstand kommt. Das Moment der Trägheit wird durch die Geschwindigkeit der initialen Bewegung bestimmt.

- „Send-to-dark-side“:
  Durch das Auflegen einer ganzen Hand auf ein Objekt wird dieses nach einer Sekunde Wartezeit auf die gegenüberliegende Seite des Displays verschoben.

- „Circular menu“:
  Durch das Auflegen beider Handflächen wird ein kreisförmiges Auswahlmenü angezeigt. Die einzelnen Menüpunkte können durch eine Rotation der Handflächen ausgewählt und durch das Beenden der Berührung aktiviert werden.

- „Tether“:
  Diese Funktion kommt in Verbindung mit 360°-Darstellungen (Erddarstellungen, Panorambilder, etc.) zum Einsatz. Die von einem Benutzer vorgenommenen Änderungen der Rotation werden nach Beendigung der Berührung schrittweise wieder rückgängig gemacht.

Um Informationen zur Nutzerfreundlichkeit der entwickelten Steuerung zu erhalten wurden, einzelne „Sphere“-Exemplare in Rahmen von gut besuchten Ausstellungen eingesetzt und das Nutzerverhalten aufgezeichnet und ausgewertet. Im Folgenden sollen einige der Ergeb-

nisse aus diesen Verhaltensanalysen vorgestellt werden.

Aus den gesammelten Daten der Berührungen konnte gezeigt werden, dass kein favorisierter Standort der Nutzer gegenüber der „Sphere“ existiert. Über einen längeren Beobachtungszeitraum waren die Positionen der Berührungen gleichmäßig über das gesamte Display verteilt. Außerdem wurde festgestellt, dass eine ungleichmäßige Anordnung und Ausrichtung der dargestellten Objekte die Nutzer eher dazu verleitet mit dem Display zu interagieren.

> *“We also observed that people were much more inclined to start touching and interacting with Sphere if the objects were left in a disorganized, “messy” arrangement by previous users. In contrast, when the interface was reset and displayed objects were realigned, we usually had to demonstrate and explain that Sphere is indeed touch-sensitive and interactive.“*
> [BEN-08]:9

In diesem Zusammenhang weisen BENKO et al. daraufhin, dass dieser Umstand besonders bei Installationen, die eine Nutzerinteraktion provozieren sollen, wie z.B. in Museen, berücksichtigt werden sollte. Diese Erkenntnis lässt sich vermutlich auch auf einen taktilen Hyperglobus übertragen. So könnte ein taktiler Hyperglobus so konfiguriert werden, dass er nach einer gewissen Zeit ohne Nutzereingabe eine „unnatürliche“ Rotation einnimmt, um das Publikum dazu zu verleiten den Globus zu berühren um einen als „normal“ empfundenen Zustand wiederherzustellen.

Obwohl hauptsächlich der Photo- und Videobrowser eingesetzt wurde, fanden auch Versuche mit 360°-Datensätzen (Erddarstellungen und Panoramavideos) statt. Diese Darstellungen konnten von den Nutzern durch Berührung und Bewegung frei rotiert werden. Um Konflikte im Mehrbenutzermodus zu vermeiden, wurde die Kontrolle über die Rotation bei konkurrierenden Berührungen immer derjenigen zugesprochen, die als Erste registriert wurde. Allerdings sorgte diese Funktion für Irritationen bei denjenigen Benutzern, die keine Kontrolle über die Darstellung hatten. Deshalb entschlossen sich die Entwickler eine weitere Funktion mit dem Namen „tether“ (engl: anketten) zu implementieren. Diese sorgt dafür, dass nach der Beendigung einer Nutzeraktion die durchgeführten Manipulationen rückgängig gemacht werden und das dargestellte Bild wieder zu einer „natürlichen“ Ausrichtung zurückkehrt. Dieses Verhalten auf einen taktilen Hyperglobus zu übertragen, stellt technisch eine Herausforderung dar, da ab einem gewissen Globendurchmesser größere Veränderungen der Rotation durch mehrere aufeinander folgende Aktionen durchgeführt werden. Eine nicht durch

den Benutzer hervorgerufene, softwaregesteuerte Veränderung der Rotation des Globus zwischen diesen Einzelaktionen ist bei solchen Fällen nicht erwünscht. Darüber hinaus treten bei der Kombination von mehreren Rotationen weitere Effekte auf, die in Kapitel 5 genauer betrachtet werden. Zusammenfassend kann festgestellt werden, dass nicht alle Ergebnisse die BENKO et al. durch ihre Untersuchungen gewonnen haben eins zu eins auf einen taktilen Hyperglobus übertragbar sind. Der eingesetzte Bild- und Videobrowser stellt andere Anforderungen an die Benutzerschnittstelle als ein taktiler Hyperglobus, der zur Darstellung von Geoinformationen auf globaler Ebene benutzt wird. Dennoch existieren Überlappungsbereiche zwischen den beiden Anwendungssituationen, so dass die Resultate als Grundlage für weitere Untersuchungen benutzt werden können.

4.2 HUMAN MEDIA LABS

„Human Media Labs“ ist der Name einer Forschungsabteilung, die zur Queen's University in Kingston (Kanada) gehört. Im Jahr 2011 veröffentlichten BOLTON et al. ein Paper, in dem sie die Ergebnisse ihrer Arbeit an einem selbst entworfenen, berührungsempfindlichen sphärischen Display („SnowGlobe“) präsentieren (siehe [BOL-11]). Im Gegensatz zum oben beschriebenen Projekt von Microsoft wurden hier keine kommerziellen Displays benutzt, sondern es wurde versucht ein möglichst kostengünstiges System im Eigenbauverfahren herzustellen. Konkrete Angaben über die Kosten ihres Systems liefern die Autoren zwar nicht, aber die in [HML-11] angeführten einzelnen Komponenten des „SnowGlobe“ lassen vermuten, dass sie nur einen Bruchteil der Kosten eines kommerziellen Globus betragen. Zum Vergleich sei hier erwähnt, dass gleichartige Systeme von verschiedenen Firmen für ca. 50.000 € angeboten werden.

Dieses Display entspricht einem taktilen Hyperglobus mit einer spiegelbasierten Innenprojektion mit einem Durchmesser von ca. 90cm (36 Zoll). Um eine kontrastreiche Darstellung des Bildes zu ermöglichen, wurde die Innenseite des Globenkörpers durch ein Sandstrahlverfahren aufgeraut, wodurch auf eine (mutmaßlich teurere) spezielle Beschichtung verzichtet werden konnte. Die Touchfunktionalität basiert auf dem Prinzip der diffusen Illumination. Als Sensoren werden zwei Videokameras verwendet, deren Bilder jeweils eine Hemisphäre abdecken. Zusätzlich wurde ein Motiontracking-System eingesetzt über dessen acht Kameras die Position des Globus und die der Benutzer vom Computer erfasst werden kann.

Diese Konfiguration ermöglicht die pseudo-voluminetrische Darstellung von dreidimensionalen Objekten auf einem sphärischen Display. Dabei wird ein vom Standpunkt des Benutzers abhängiges Bild eines Objekts dargestellt. Dadurch erhält ein Nutzer, der sich um das Display bewegt, den Eindruck, dass das dargestellte Objekt sich im Zentrum des Displays befindet und sich nicht bewegt. Im Vergleich mit der Microsoft „Sphere“ ist der Umfang der Steuerungsbefehle die durch Berührungen eingegeben werden können sehr gering und umfasst folgende drei Funktionen:

- „Rotation“:
  Um das auf dem Display abgebildete Objekt zu rotieren, wird eine einfache Berührung und Bewegung ausgewertet. Die Rotation ist dabei auf zwei starre Achsen beschränkt, so dass keine beliebigen Rotationen durchgeführt werden können.

- „Scaling“:
  Ähnlich dem Verhalten der Microsoft „Sphere“, wird die Änderung der Distanz zwischen

zwei Berührungspunkten auf die Größe des abgebildeten Objekts übertragen.

- „Swiping“:
  Durch eine genügend schnelle Bewegung wird zum nächsten bzw. vorherigen Objekt gewechselt.

Das Konzept, eine Rotation auf bestimmte Achsen zu beschränken, ist unter Umständen auch für einen taktilen Hyperglobus sinnvoll, da wie bereits erwähnt, Rotationen von dreidimensionalen Körpern auf verschiedene Arten umgesetzt werden können und einige der möglichen Lösungen Benutzer verwirren könnten.

Abbilung 16: Human Media Labs, Durchführung einer "Scaling"-Geste an einem "Snow Globe" (Quelle: [BOL-11]:4)

Für das Vorhaben des Autors einen taktilen Hyperglobus zu modifizieren und mit einem Touchscreen-System zu versehen, ist die Tatsache, dass BOLTON et al. ein ähnliches Projekt ausschließlich mit billigen und leicht verfügbaren Komponenten umgesetzt haben, eine Bestätigung für den geplanten Einsatz eines Systems auf Basis der diffusen Illumination.

4.3 PUFFERFISH

Die Firma Pufferfish (Edinburgh, England) ist ein Anbieter, der sich auf sphärische Displays spezialisiert hat. Auch ein berührungsempfindliches Display findet sich in ihrem Angebot (vgl. [PUF-12]). Allerdings veröffentlicht der Hersteller keine technischen Details oder genaue Spezifikationen zu der eingesetzten Touch-Technologie. Aus diversen Broschüren und Werbevideos lässt sich aber schließen, dass die Displays über eine Innenprojektion mittels eines Fischaugenobjektivs verfügen. Der Durchmesser des größten angebotenen Displays beträgt 3,5 m. Zudem sind neben starren, beschichteten Kunststoffkugeln Varianten mit einem aufblasbaren Globenkörper verfügbar, die einige Vorteile im Bezug auf die Transportfähigkeit des Systems aufweisen. Bei der Durchsicht des auf der Homepage des Anbieters verfügbaren Materials, finden sich nur vereinzelt Anwendungen, die das sphärische Display zur Anzeige globaler Datensätze nutzen. Überwiegend, so scheint es, wird es zur Darstellung künstlerischer Visualisierungen und als Blickfang eingesetzt.

Abbildung 17: Der Schauspieler Will Smith bei der Bedienung eines touchfähigen sphärischen Displays der Firma Pufferfish (Quelle: [PUF-12])

5. tTHG - Eigene Umsetzung

5.1 AUSGANGSSITUATION

Die Arbeitsgruppe Kartographie und Geoinformation, die am Institut für Geographie und Regionalforschung (IFGR) der Universität Wien tätig ist, beschäftigt sich schon seit mehr als 10 Jahren mit der Erforschung und Weiterentwicklung von Globen. Unter der Leitung von Ass.-Prof. Mag. Dr. Andreas Riedl wurden seit 1999 acht verschiedene Forschungsprojekte, die einen Bezug zum Thema Globus aufweisen, durchgeführt. Um auf die neuen Technologien in diesem Bereich zu reagieren und die Globenforschung auf diesem Feld fortzuführen, wurde 2005 schließlich die „Hyperglobe Research Group“ (HRG) ins Leben gerufen und als erste europäische Forschungseinrichtung erhielt das IFGR ein sphärisches Display (siehe [RIE-10]).

> *„Sie* [Anmerkung: gemeint ist die HRG] *verfolgt das Ziel, den taktilen Hyperglobus zu einem Instrument der Wissensvermittlung globaler Sachverhalte zu machen.“*
> [RIE-10]:8

Der Autor arbeitet seit 2007 aktiv in der HRG mit und beschäftigt sich in diesem Zusammenhang unter anderem mit der Programmierung einer 3D-Software zur Darstellung von Inhalten auf sphärischen Displays. Darüber hinaus ist er als freier Mitarbeiter für die Firma Globoccess (Hamburg) tätig, die taktile Hypergloben verkauft und Serviceleistungen rund um ihre Produkte anbietet. Das Aufgabenfeld umfasst dabei vor allem die Installation von Globen vor Ort und die Einschulung von Personen in die Bedienung. Überwiegend werden diese Installationen und Schulungen in Museen, Science-Centren und Ausstellungen durchgeführt, wie z.B. im Deutschen Museum (München, Deutschland), Technorama (Winthertur, Schweiz), Vulcania (Roches, Frankreich) und „The World EXPO 2010“ (Shanghai, China). Im Rahmen dieser Tätigkeiten konnte der Autor durch den Kontakt mit den Globensystemen und -nutzern aller Ebenen (Betreiber, Content-Ersteller, Techniker, Besucher) zahlreiche Erfahrungen und Eindrücke sammeln, die natürlich in dieses Projekt mit eingeflossen sind.

5.1.1 HARDWARE

Im Jahr 2006 wurde am Institut für Geographie und Regionalforschung der Universität Wien der europaweit erste Hyperglobe an einer Forschungseinrichtung installiert. Es handelt sich dabei um ein Modell der Firma ArcScience mit der Bezeichnung Omniglobe (siehe Abbildungen 18, 19 und 20). Dieser Globus basiert auf einem spiegelbasierten Inneprojektionssystem. Der Durchmesser des Globenkörpers beträgt ca. 1,5m. Er besteht aus zwei miteinander verklebten Kunststoffhalbschalen. Eine Beschichtung auf der Außenseite sorgt für eine kontrastreiche Darstellung der projizierten Bilder. Am oberen Ende des Globenkörpers befindet sich eine Befestigungsvorrichtung, über die der Disperser mit dem Globus verbunden ist. Eine Besonderheit ist die sogenannte Ghost-trap. Dabei handelt es sich um einen ca. 50cm langen Hohlzylinder aus Metall. Dieser befindet sich Inneren und ist am Übergang zwischen Gehäuse und Globenkörper befestigt. Er dient dazu, unerwünschte Reflexionen zu verhindern, die das projizierte Bild beeinträchtigen würden. Der Videoprojektor, der innerhalb des Gehäuses untergebracht ist, verfügt über eine maximale Auflösung von 1400 x 1050 Pixeln. Um den horizontal projizierten Lichtstrahl des Projektors in Richtung des Disperser umzulenken, ist ein zusätzlicher flacher Spiegel im Gehäuse angebracht. Da während des Betriebs dieses Projektors eine nicht unerhebliche Wärmemenge abgegeben wird, befindet sich im Gehäuse ein Lüfter in Kombination mit einem Staubfilter, um diese Wärme abzutransportieren. Der Projektor ist über ein DVI-Kabel mit einer Workstation verbunden. Die darauf ausgeführte Software zum Betrieb des Globus wird im nächsten Abschnitt vorgestellt.

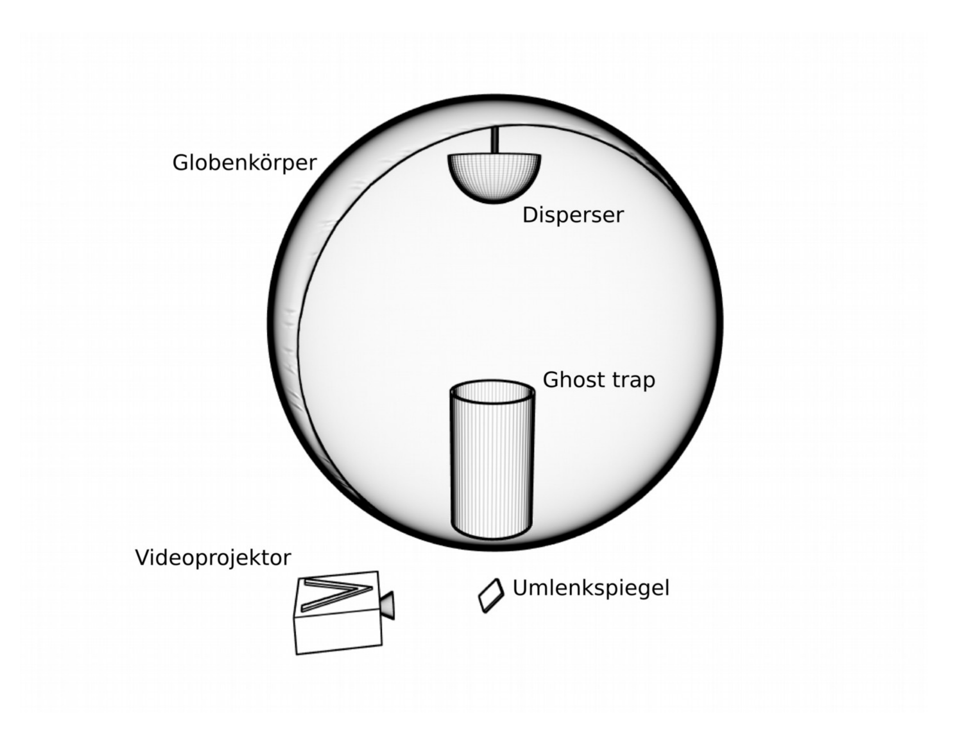


Abbildung 18: ArcSciene "OmniGlobe", schematischer Aufbau (Quelle: eigene Darstellung)

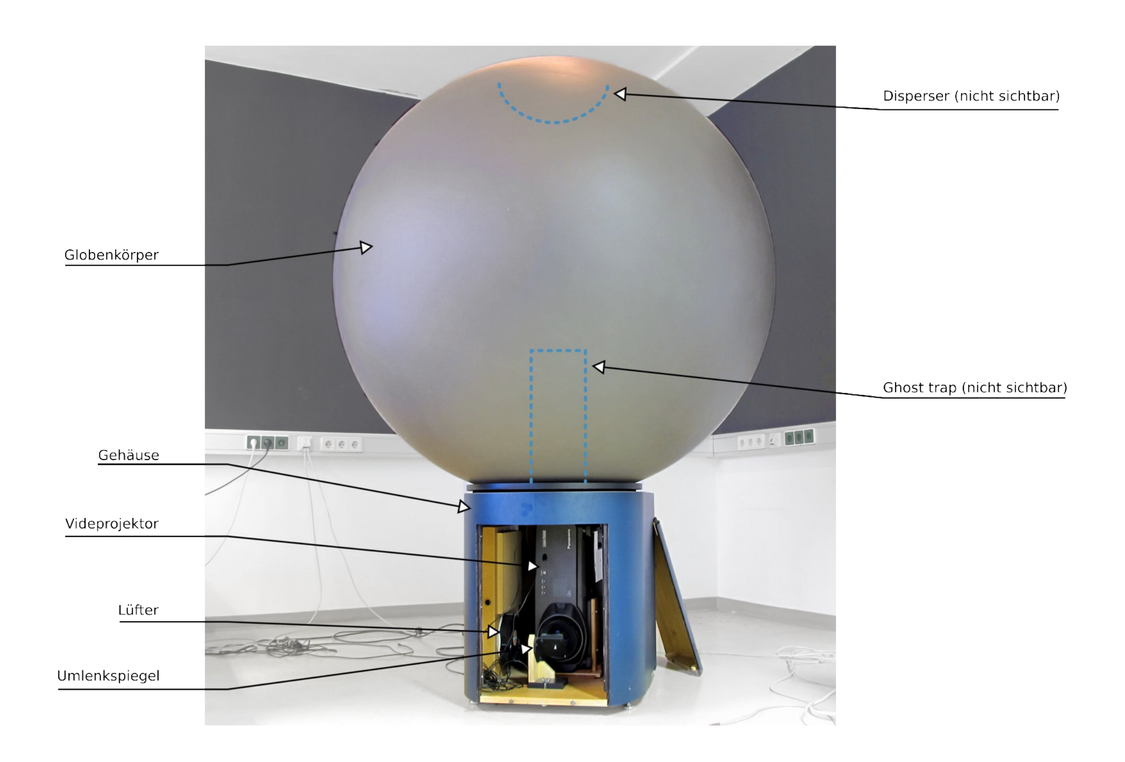


Abbildung 19: OmniGlobe, Institut für Geographie und Regionalforschung, Universität Wien (Quelle: eigene Darstellung)

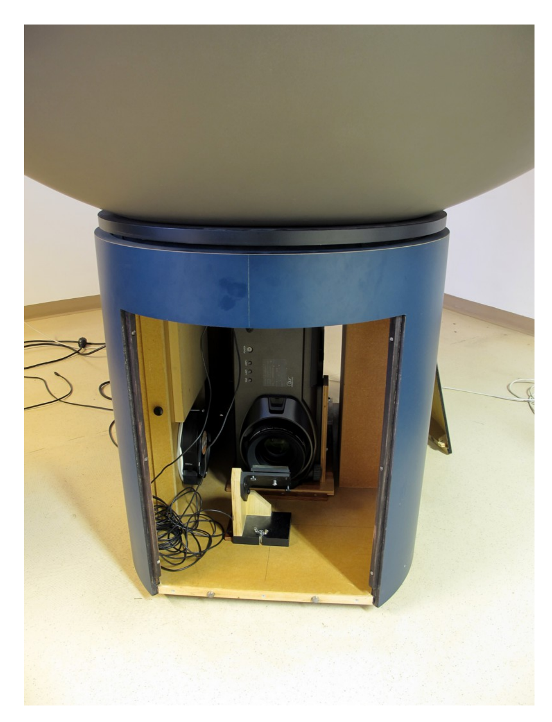


Abbildung 20: OmniGlobe, Institut für Geographie und Regionalforschung, Universität Wien, Detailaufnahme (Quelle: eigene Aufnahme)

5.1.2 SOFTWARE

Seit 2008 arbeitet die HRG an dem Projekt OmniSuite (OS) mit dem Ziel ein gleichnamiges Softwarepaket zu erstellen, welches sämtliche Funktionen zur Verfügung stellt, die für die Aufbereitung und Darstellung von globalen Sachverhalten an einem taktilen Hyperglobus benötigt werden. Folgende Projektbeschreibung befindet sich auf der Homepage der HRG:

> *„Gegenstand des Projekts ist die Konzipierung und Umsetzung eines plattformunabhängigen Authoring- und Präsentationssystems zur Geovisualisierung globaler Sachverhalte auf taktilen Hypergloben. Dies reicht von der Implementierung einer geeigneten 3D-Engine über die Entwicklung eines Framework-basierten User-Interface bis hin zur 'on the fly' Projektion globaler (dynamischer) Sachverhalte auf Basis von Echtzeitdaten.“*
> [IFGR-12]

Zu beachten ist, dass der Begriff „plattformunabhängig“ sich hier nicht auf das Betriebssystem eines Computers bezieht, sondern dass die Software auf allen verfügbaren Typen von taktilen Hypergloben eingesetzt werden kann. Da diese oben genannten Anforderungen an die Software ein sehr breites Spektrum umfassen, wurde sie modular konzipiert: Es wurden verschiedene Einzelanwendungen entwickelt, welche jeweils einen bestimmten Aufgaben- und Funktionsbereich abdecken. Im Folgenden werden die einzelnen Module, die OS beinhaltet, kurz vorgestellt und ihre wichtigsten Funktionen erläutert. Eine wesentlich umfangreichere Beschreibung sowie Hintergrundwissen zum Thema „3D Graphik und Programmierung am Hyperglobus“ findet sich in der Diplomarbeit von KRISTEN (siehe [KRI-12]), der als Mitarbeiter der HRG für die Programmierung der OS-Anwendungen zuständig ist.

Um das Verständnis zu erleichtern, muss vorab erwähnt werden, dass die für die Darstellung am Globus gedachten Inhalte im Umfeld der OS als „Stories“ bzw. „global Stories“ bezeichnet werden (siehe [HRU-09] und [RIE-11]). Das zentrale Element bildet ein oder mehrere Materialien. Diese bestehen aus einem einzelnen Bild oder Bildsequenzen. Zusammen mit Angaben über die zeitliche Abfolge der verschiedenen Materialien und Rotationsvorgänge, Audiodaten, Zusatzinformationen und weiteren Ebenen, z.B. für die Darstellung einer Legende, ergibt sich daraus eine Story. Thematisch zusammenhängende Stories können wiederum zu „Collections“ gebündelt werden (z.B könnte eine Collection „Sonnensystem“ eine Story zu jedem einzelnen Planeten enthalten).

Die verschiedenen Module lassen sich zu den Kategorien Präsentation und Authoring, also die Darstellung von Stories bzw. die Erstellung, Aufbereitung und Verwaltung derselben zusammenfassen (siehe Abbildung 21). Zusätzlich können in einem gesonderten Modul alle Einstellungen und Konfigurationsdateien bearbeitet werden.

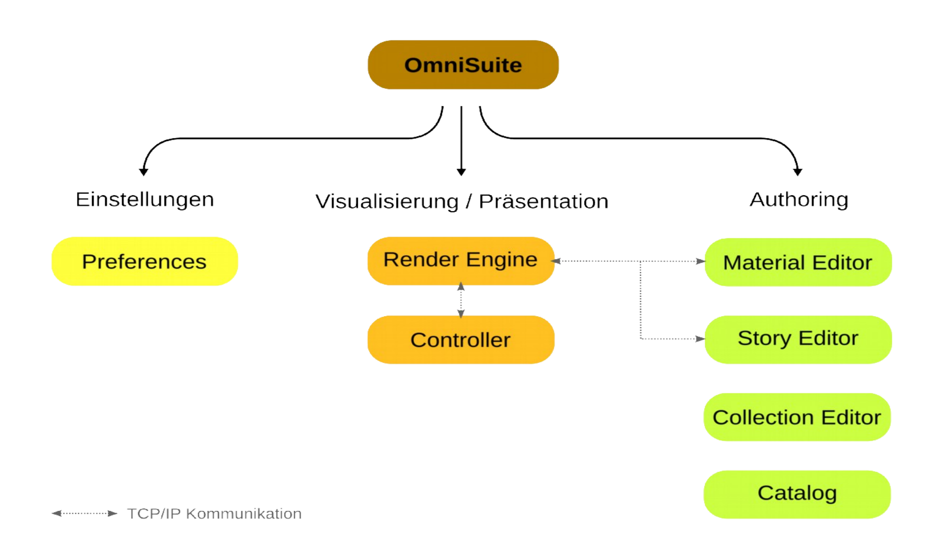


Abbildung 21: OmniSuite, Übersicht über die einzelnen Module (Quelle: eigene Darstellung)

5.1.2.1 Visualisierung und Präsentation

Das zentrale Modul wird als Render Engine (RE) bezeichnet. Innerhalb dieses Moduls erfolgt die Umwandlung von Geodaten in eine für die Darstellung am Globus passende Projektion. Die Ausgangsdaten bilden quadratische Plattkarten. Die Umprojektion ergibt sich durch die Umkehrung des Strahlengangs an einem virtuell konstruierten Globus mit Hilfe der frei verfügbaren 3D-Renderengine OGRE. Während am realen Globus das Bild vom Beamer über Spiegel auf die Kugel projiziert wird, wird in der RE das Ausgangsmaterial auf den virtuellen Globenkörper gemappt[1]. Anstelle des Beamers wird in der virtuellen Umgebung eine Kamera eingesetzt und auf den konvexen Spiegel innerhalb des Globus gerichtet. Dieser Spiegel zeigt die Reflexion des auf den Globenkörper dargestellten Bildmaterials. Das durch die Kamera aufgenommene Bild besitzt nun die für eine korrekte Darstellung am Globus notwendige Projektion und kann als Eingangsignal an den realen Projektor übertragen werden. Abbildung 22

1) „Mapping" ist die Bezeichnung für eine Technik zur Abbildung von zweidimensionalen Texturen bzw. Bildern auf dreidimensionale Objekte. Details dazu finden sich bei [KRI-12] in Kapitel 4.10

zeigt ein solches Bild. Zu erkennen ist, dass es sich um eine Azimutalprojektion handelt. Befindet sich der Globus in der Standardposition (keine Rotation), wird der Südpol durch einen Punkt in der Bildmitte verkörpert. Der Nordpol sowie sämtliche Breitenkreise werden zu konzentrischen Kreisen umgewandelt, deren Mittelpunkt in der Bildmitte liegt. Meridiane werden durch gerade Linien dargestellt, die durch das Bildzentrum verlaufen.

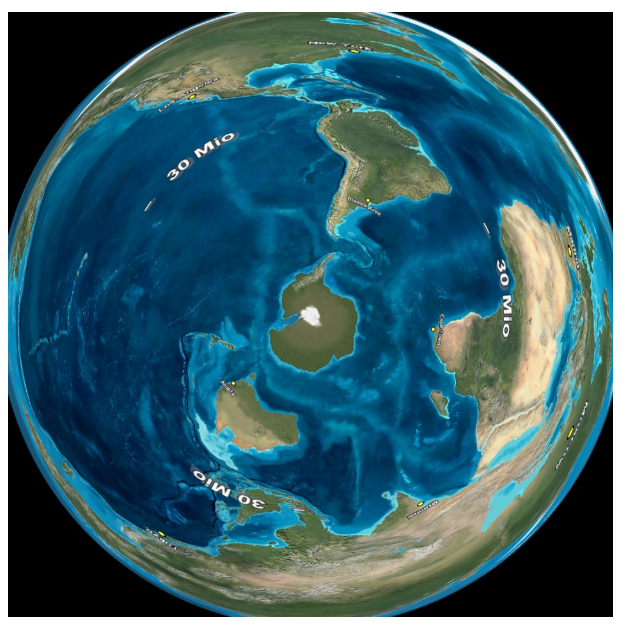


Abbildung 22: Screenshot, Render Engine, projizierte Plattkarte (Quelle: OS, eigene Darstellung)

Das Controller-Modul stellt eine Bedienoberfläche für die Steuerung bereit. Üblicherweise wird diese Oberfläche auf einem Bildschirm angezeigt, der sich in unmittelbarer Nähe zum Globus befindet. Eine solche Anordnung ist insofern von Vorteil, da der Nutzer die Reaktion auf Eingaben direkt wahrnehmen kann. Zur Bedienung werden neben Tastatur und Maus häufig auch Touchscreens eingesetzt. Die Programmoberfläche ist in einzelne Bereiche (Frames) aufgeteilt, die jeweils einen Aspekt der Globenfunktionalität abdecken. Über eine Konfigurationsdatei (bzw. über das Modul Preferences) kann die Gestaltung und das Layout der Frames an die Bedürfnisse der Nutzer angepasst werden. Oft wird dies bei Globen verwendet, die der Öffentlichkeit oder Besuchern zugänglich sind. So kann durch das deaktivieren einzelner Frames verhindert werden, dass die Anwendung mutwillig oder versehentlich geschlossen wird. Die Funktionsbereiche der fünf Frames sind wie folgt aufgeteilt (vgl. Abbildung 23):

1. Zeitleiste, Sprachauswahl

2. Übersicht und Auswahl der vorhandenen Stories
3. Themensammlungen und Schnellvorschau
4. Zusatzinformationen zur aktiven Story
5. Manipulation der Rotation des Globus
6. Übersicht und Auswahl der Lesezeichen
7. Auswahl der verschiedenen Ebenen der aktiven Story

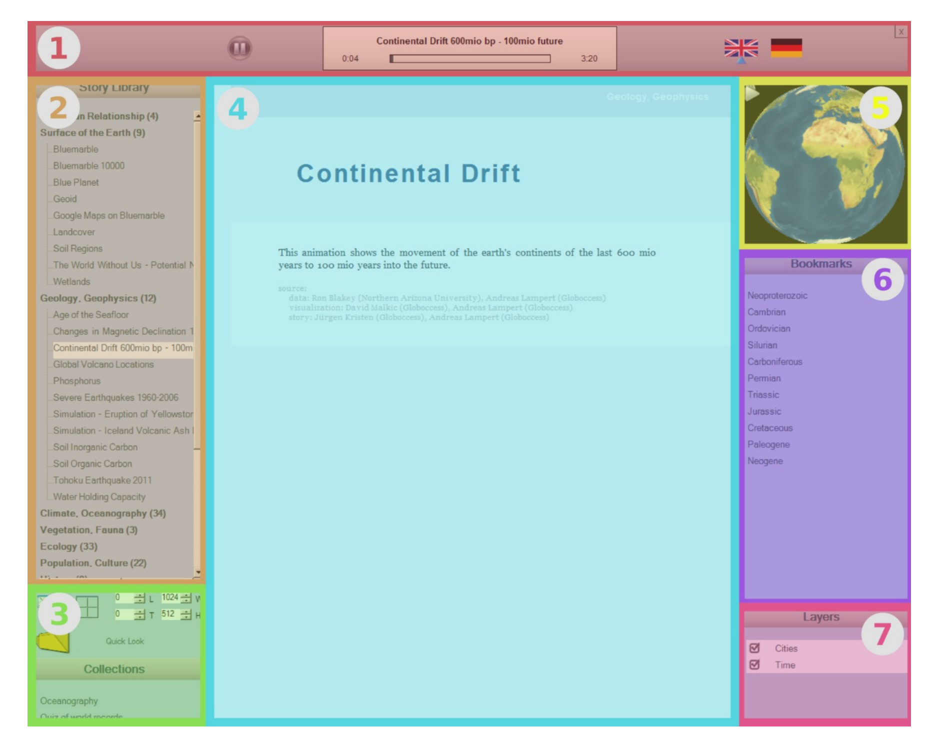


Abbildung 23: OS Modul "Controller", Screenshot mit Frames (Quelle: eigene Darstellung)

Da vom Controller nur die Oberfläche für Eingaben erzeugt wird und die Bilderzeugung für den Globus allein von der RE übernommen wird, müssen diese beiden Module miteinander kommunizieren. Dies geschieht in Form von speziellen Textnachrichten (API[2]), die über eine Netzwerkschnittstelle ausgetauscht werden. Diese Kommunikation kann prinzipiell bidirektional erfolgen, allerdings wird zum jetzigen Entwicklungsstand die RE als alleiniger Empfänger eingesetzt. Als Sender dienen die Module Controller, Story Editor und Material Editor (siehe Abbildung 21). Die Kommunikation über eine Netzwerkschnittstelle ermöglicht es,

2) API = Application Programming Interface, zu deutsch: Schnittstelle zur Anwendungsprogrammierung (vgl. [ERT-06]:3).

dass die einzelnen Module auf getrennten Rechnern ausgeführt werden können. So kann die rechenintensive RE auf einer dementsprechend leistungsstarken Workstation betrieben werden, während die Steuerung z.B. über einen tragbaren Tablett-Computer erfolgt, der über ein Funknetzwerk mit der Workstation verbunden ist. Die API ermöglicht auch den Einsatz selbsterstellter Programme zur Steuerung des Globus, bietet dabei allerdings nicht den selben Funktionsumfang wie das Controller Modul.

5.1.2.2 Authoring

Wie oben erwähnt, werden für die Erstellung einer Story die als Bilddateien vorliegenden Ausgangsdaten zu sogenannten Materialien zusammengefasst. Die dazu nötigen Informationen werden in eine Skriptsprache übersetzt und in einer Textdatei (Materialskript) abgespeichert. Da das Editieren dieses Skripts in einem Texteditor nicht nur unbequem ist, sondern auch die Gefahr besteht, dass die Validität des Skripts z.B. durch Syntaxfehler verletzt wird, beinhaltet OS das Modul Material Editor, welches eine graphische Oberfläche (GUI) zur Bearbeitung dieser Materialskripte zur Verfügung stellt. Über diese GUI können sowohl neue Materialien erstellt als auch bereits vorhandene bearbeitet werden. Zu den wichtigsten Angaben in einem Materialskript gehören der Name und Speicherort der verwendeten Bilddateien. Im

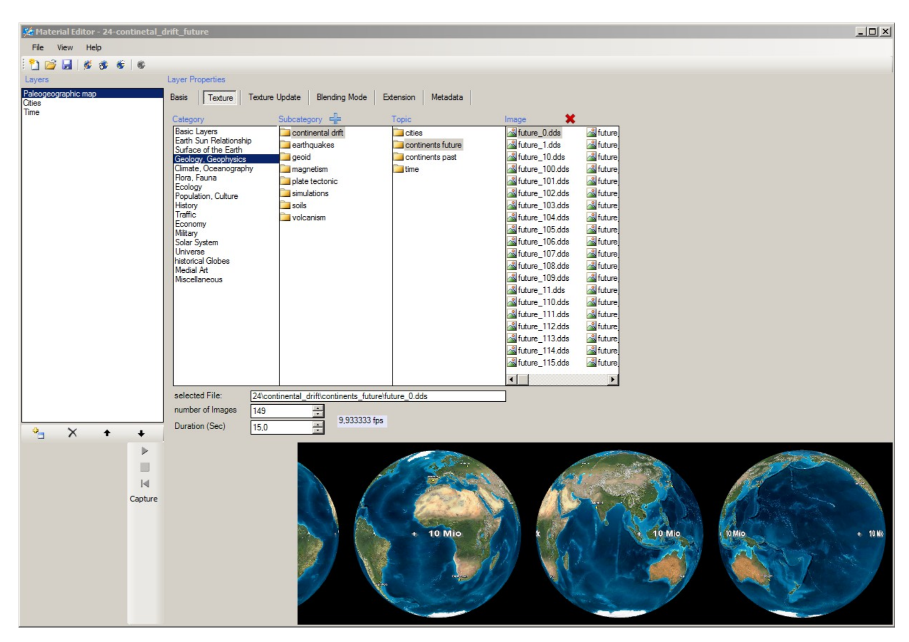


Abbildung 24: Screenshot: OS Material Editor (Quelle: OS, eigene Darstellung)

Falle von Bildsequenzen sind zusätzlich Informationen über die Dauer der Sequenz nötig, um daraus die korrekte Bildrate ermitteln zu können. Videoformate werden nicht unterstützt. Der Grund dafür ist, dass die Umprojektion innerhalb der RE in Echtzeit erfolgt und die Dekodierung eines Videos zusätzliche Rechenleistung und Speicher erfordert.

Für Bilder mit transparenten Anteilen, die über einen Alphakanal verfügen, kann zwischen verschiedenen Überblendungseffekten gewählt werden. Diese Technik wird vor allem für die Darstellung von Texten und Legenden verwendet. Abbildung 24 zeigt einen Screenshot des Material Editors.

Analog zu den oben erläuterten Materialskripten kommen auch bei der Erstellung von Stories solche Skripten und eine entsprechende GUI zum Einsatz. Die Oberfläche dieses Story Editor genannten Moduls ist mit derer von Videobearbeitungsprogrammen vergleichbar. Sie verfügt über zwei Zeitleisten (Spuren) mit deren Hilfe sich die Abfolge der Materialien und Rotationen in Abschnitte gliedern lässt (siehe Abbildung 25). Über zusätzliche Eingabefelder können die Eigenschaften dieser Abschnitte verändert werden. Für Rechner, die den Anforderungen des Echtzeitbetriebs nicht gewachsen sind, besteht außerdem die Möglichkeit eine Story als Einzelbildsequenz zu exportieren, die anschließend in eine Videodatei umgewandelt und abgespielt werden kann. Allerdings kann in diesem Fall nicht mehr mit dem Controller bzw. über die API auf die Story eingewirkt werden.

In der Regel werden die Inhalte für den Globus nicht auf denselben Rechnern erstellt bzw.

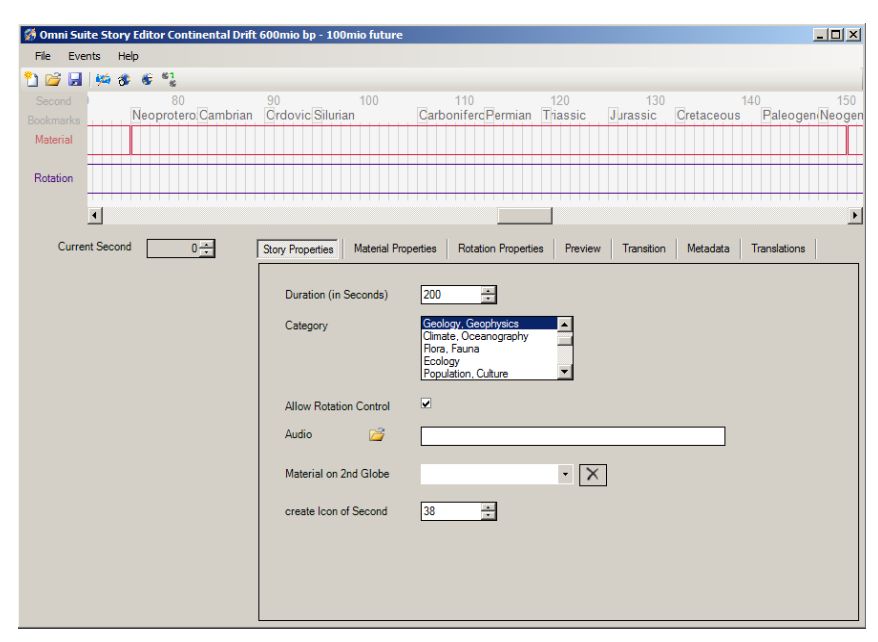


Abbildung 25: Screenshot: OS Story Editor (Quelle: OS, eigene Darstellung)

aufbereitet, die auch für die Präsentation, also den Betrieb, verwendet werden. Daher ist es notwendig die Materialien, Stories, etc. von einem Speicherort zu einem anderen übertragen zu können. Für komplexe Stories werden sehr große Datenmengen und eine große Anzahl von Dateien erzeugt. Diese sind ähnlich zu den Skripten nach bestimmten Regeln sortiert und werden in einem sogenannten Medienverzeichnis abgespeichert. Auch hier besteht die Gefahr, dass durch das manuelle Kopieren, Löschen oder Verschieben von Dateien die Integrität dieses Verzeichnisses verletzt wird. Mitunter können dann bestimmte Stories, Funktionen oder ganze Module nicht mehr korrekt wiedergegeben oder ausgeführt werden. Um diese Probleme zu vermeiden, stellt das Modul Catalog alle benötigten Funktionen zum Verwalten von Inhalten zur Verfügung. Aus Abbildung 26, einem Screenshot des Catalog Moduls, lässt sich erkennen, dass dessen GUI wie ein zweispaltiger Dateimanager aufgebaut ist. In der linken Spalte wird ein lokales Medienverzeichnis angezeigt, während das Basisverzeichnis für die rechte Spalte beliebig gewählt werden kann (lokaler Speicherort, externer Datenträger, Netzwerkverbindung, ...). Nun können einzelne Materialien, Stories oder Collections zwischen diesen beiden Speicherorten kopiert, verschoben oder gelöscht werden. Über die Funktion Webcatalog können Inhalte auch über das Internet bezogen und aktualisiert werden.

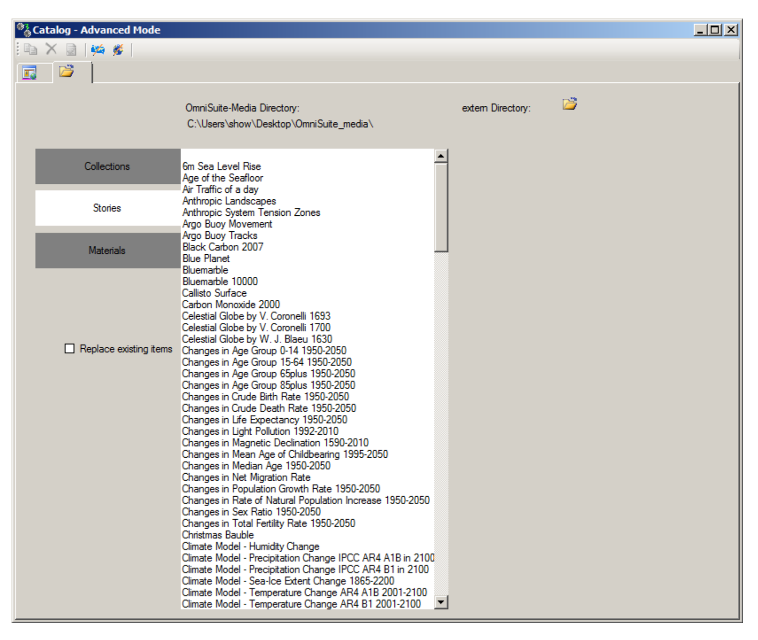


Abbildung 26: Screenshot: OS Catalog (Quelle: OS, eigene Darstellung)

5.2 KONZEPT

Nachdem alle nötigen Grundlagen und Voraussetzungen für die Entwicklung eines touchfähigen THG (tTHG) in den vorherigen Kapiteln behandelt wurden, kann jetzt ein Konzept für die Realisierung verfasst werden.

Der am Institut vorhanden THG ist der einzige Globus auf den ein Zugriff möglich ist und daher soll dieser für die Umsetzung verwendet werden. Eine Bedingung dafür ist jedoch, dass keine permanenten Veränderungen durchgeführt werden. Alle Modifikationen müssen vollständig rückgängig gemacht werden können und die Struktur sowie sämtliche Bestandteile des Globus müssen unverändert bleiben.

Aufgrund dieser Voraussetzungen sowie der im dritten Kapitel gewonnenen Erkenntnisse soll die Touchfunktionalität über das Prinzip der diffusen Illumination erreicht werden. Alle anderen vorgestellten Methoden würden Umbauten erfordern, die nicht mit den vorher erwähnten Bedingungen vereinbar wären:

- Systeme die auf elektrischer oder akustischer Sensorik beruhen, setzen eine aufwendige Modifikation des Globenkörpers voraus.
- Im Bereich der optischen Sensorik scheidet die Leuchtgittermethode aus, da sie nicht auf gewölbten Oberflächen eingesetzt werden kann.
- Die für die Installation eines FTIR-Systems erforderliche Montage der LEDs an einer Kante oder Nahtstelle des Globenkörpers ist ebenfalls nicht ohne Modifikationen möglich.

Daher müssen am Globus eine oder mehrere Infrarotkameras als Sensoren sowie IR-Leuchtquellen als Emitter montiert werden. Diese müssen so positioniert werden, dass das IR-Licht im Inneren des Kugel ausgesendet wird und die Videokameras die durch eine Berührung hervorgerufenen Reflexion erfassen können. Die Touch-Funktionalität soll möglichst auf der gesamten Globenoberfläche zur Verfügung stehen. Um dies zu gewährleisten, muss ein möglichst großer Bereich des Globenkörpers durch eine oder mehrere Kameras erfasst werden. Dabei sind verschiedene Optionen denkbar:

- Mehrere Kameras innerhalb des Globus:
  Innerhalb des Globenkörpers können mehrere Kameras am Fuße der Ghost-trap platziert werden. Das Blickfeld jeder dieser Kameras erstreckt sich über einen Teil des Globus. Dabei kommt es zu Überlappung zwischen den benachbarten Kameras.
- Eine Kamera im Inneren mit Fischaugenobjektiv:

Eine Variante, die auf eine einzelne Kamera mit Fischaugenobjektiv zurückgreift, kann auf Grund der Ghost-trap nicht umgesetzt werden, da entweder nur ein Teilbereich des Globenkörpers im Blickfeld liegen oder die Kamera den Strahlengang des Projektors stören und dadurch Abschattungen auf der Globusoberfläche hervorrufen würde.

- Eine Kamera im Gehäuse bzw. Sockel:
  Eine weitere Möglichkeit stellt die Verwendung einer einzelnen Kamera dar, die neben dem Umlenkspiegel platziert und auf den Disperser gerichtet ist. Sofern der Disperser gänzlich innerhalb des Kamerablickfelds liegt, kann die gesamte Globusoberfläche aufgenommen werden.

Da die Positionierung und Auswertung der Bilddaten mehrerer Kameras ungleich aufwendiger ist als bei einer einzigen Kamera, wird die letzte Variante bevorzugt. Als Leuchtquellen sollen Infrarot-Leuchtdioden (IR-LED) verwendet werden. Diese beiden Komponenten - Kamera und Dioden - können relativ kostengünstig beschafft werden. HALBRITTER weist in [HAL-10]:6 drauf hin, dass die Beleuchtungssituation möglichst gleichmäßig und homogen sein sollte. Daher scheint der Einsatz von mehreren Dioden und eine Platzierung an der Ghost-trap sinnvoll.

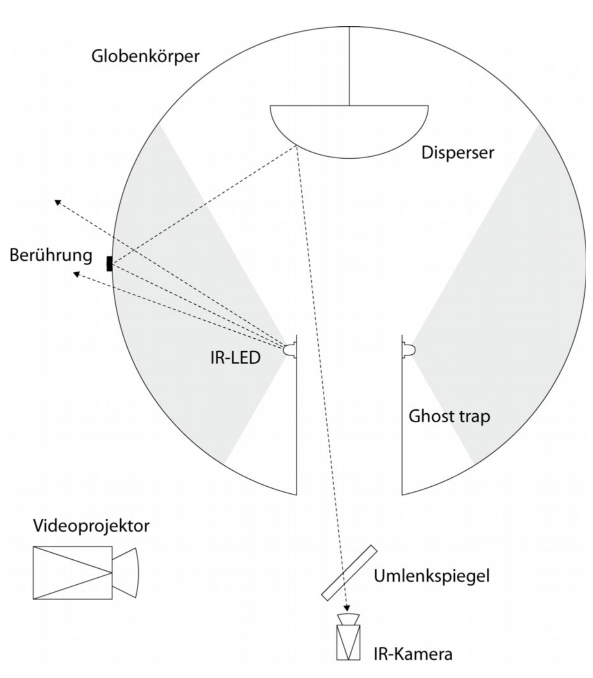


Abbildung 27: IR-Strahlengang, schematische Darstellung (Quelle: eigene Darstellung)

Aus Abbildung 27 geht der Strahlengang des IR-Lichts hervor: Die IR-LEDs befinden sich am oberen Ende der Ghost-trap und leuchten den Innenraum des Globenkörpers aus. Das IR-

Licht kann den Globenkörper im Normalfall ungehindert passieren. Bei einer Berührung von außen erfolgt jedoch eine Reflexion. Ein Teil dieses reflektierten Lichts gelangt zum Disperser und schließlich zur IR-Kamera.

Zur Verarbeitung und Auswertung der gewonnenen Bilddaten soll eine spezielle Software entwickelt werden (im Weiteren als tTHG-Software bezeichnet). Als Programmiersprache wird C++ verwendet. Da die Bildverarbeitung ein sehr komplexer Vorgang ist, soll diese soweit möglich mit Hilfe von frei verfügbaren und quelloffenen Software-Bibliotheken durchgeführt werden. Um die Resultate der Bildverarbeitung in Aktionen am Globus umzusetzen, ist eine Kommunikation mit dem RE Modul nötig. Diese soll über eine Netzwerkschnittstelle und der von OS zur Verfügung gestellten API erfolgen.

Das Ziel der Umsetzung ist es, an Hand eines Prototyps die prinzipielle Machbarkeit zu demonstrieren. Implementiert werden sollen eine einfache Geste zum Rotieren des Globus und eine Multitouch-Geste. Die Funktionen und Routinen, die dazu innerhalb der Software erstellt werden, sollen so beschaffen sein, dass sie als Basis zur Lösung komplexerer Aufgaben und Probleme im Zuge weitergehender Untersuchungen verwendet werden können.

5.3 ZUSÄTZLICH ERFORDERLICHE KOMPONENTEN UND TECHNIKEN

Aus dem Konzept für die Entwicklung eines tTHG geht hervor, dass neben dem am Institut vorhandenen Globus weitere Elemente für die Realisierung notwendig sind. Es handelt sich dabei um die beiden Hardwarekomponenten IR-LED und IR-Kamera sowie verschiedene Techniken der Bildverarbeitung und dazugehörige Software-Bibliotheken.

In diesem Kapitel wird auf deren allgemeinen Aufbau und Funktionsweise eingegangen und dargelegt, aus welchen Gründen die spezifischen Komponenten ausgewählt wurden.

5.3.1 HARDWARE

5.3.1.1 LED und IR-LED

LENK schildert im einführenden Kapitel in [LEN-11] die physikalischen Prinzipien und elektronischen Eigenschaften, die zum Verständnis der LED-Technologie nötig sind. Die Abkürzung LED steht für „light-emitting diode“ (deutsch: Licht emittierende Diode). Es handelt sich demnach um optoelektronische Halbleiterbauteile, die Licht abstrahlen, sobald sie von Strom durchflossen werden. Wie bei allen Dioden ist der Stromfluss durch eine LED nur in eine Richtung möglich.

Herkömmliche Glühbirnen emittieren Licht, das nahezu den vollständigen, vom menschlichen Auge wahrnehmbaren Frequenzbereich umfasst und daher als weißes Licht wahrgenommen wird. LEDs dagegen geben Licht nur in einem sehr schmalen Wellenlängenbereich ab und erscheinen daher farbig. Dieser Wellenlängenbereich wird vom verwendeten Halbleitermaterial bestimmt. Im Falle von IR-LEDs kommt Galliumarsenid bzw. Alluminiumgalliumarsenid zum Einsatz. Dadurch geben sie Licht im Bereich von 850nm bzw. 940nm ab (siehe Abbildung 28).

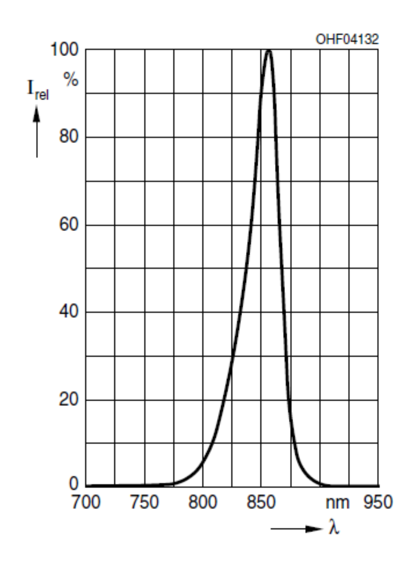
LEDs werden nach ihrer Leistungsaufnahme in Standard-LEDs und High-Power-LEDs unterschieden. Um diese Leistungsaufnahme zu bestimmen, müssen zwei Kenngrößen einer LED bekannt sein:

Zum einen die Spannung, die an der LED im Betrieb abfällt. Diese wird als Durchlassspannung

(engl: Forward voltage) bezeichnet und ist, wie die Wellenlänge des abgegebenen Lichts, ebenfalls vom verbauten Halbleitermaterial abhängig.

Zum anderen die Stärke des Stroms, der durch die LED fließt. Rote Standard-LEDs werden z.B. in Laptops oder TV-Geräten als Statusleuchten eingesetzt. Typischerweise werden sie mit einer Stromstärke von 20mA betrieben. Bei einer Durchlassspannung von ca. 2V ergibt sich daraus eine Leistungsaufnahme von 20mA * 2V = 40mW. High-Power-LEDs wurden in den letzten Jahren stark weiterentwickelt und werden im Bereich zwischen 350mA und 1A betrieben. Manche LEDs können eine Leistung von bis zu 3W oder mehr aufnehmen. Solche LEDs werden z.B. in Autoscheinwerfern oder zur Straßenbeleuchtung eingesetzt.

Im Gegensatz zu den Standard-LEDs ist hier eine zumindest passive Kühlung durch einen Kühlkörper notwendig, da die im Betrieb entstehende Wärme ansonsten das Bauteil zerstören würde. Eine weitere Eigenheit der LEDs ist die Art der erforderlichen Stromversorgung. Im Gegensatz zu Glühbirnen, die eine konstante Spannung erfordern, benötigen sie eine Energiequelle, die eine konstante Stromstärke liefert. Der Grund hierfür ist, dass der Innenwiderstand einer LED mit zunehmender Temperatur absinkt. Bei konstanter Spannung hat dies einen Anstieg der Stromstärke und eine eventuelle eine Überbelastung der LED zur Folge, welche wiederum zur vorschnellen Alterung oder zur Zerstörung des Bauteils führen kann. Abbildung 28 zeigt den Zusammenhang zwischen Stromstärke und Durchlassspannung einer IR-LED.



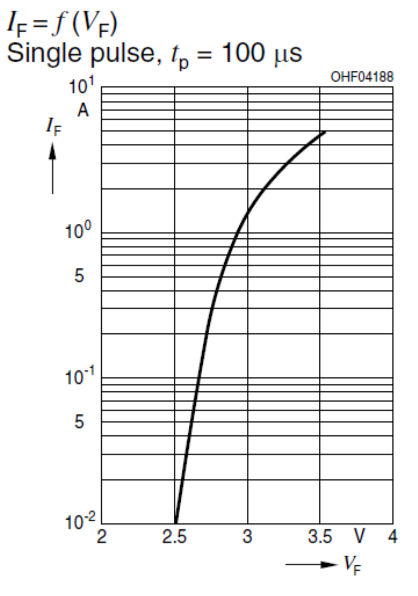


Abbildung 28: Relative spektrale Emission und Verhältnis von Stromstärke zur Spannung einer IR-LED, Typ Osram SFH 4235 (Quelle: [OSR-12]:5)

Neben der Wellenlänge des ausgestrahlten Lichts ist für einen Einsatz am tTHG die Abstrahlcharakteristik einer LED von entscheidender Bedeutung. Diese Charakteristik beschreibt in welche Richtung und in welcher Intensität das Licht von der Chipfläche[3] der LED abgeben wird.

Wie bereits im Konzept (Kapitel 5.2) erwähnt, soll der Globenkörper möglichst gleichmäßig ausgeleuchtet werden. Daher wurde ein IR-LED Typ der Firma OSRAM Seminconductors mit der Bezeichnung „SFH 4235 - Platinum Dragon“ gewählt. Diese LEDs weisen im Vergleich zu Standard-LEDs einen größeren Abstrahlbereich auf. Abbildung 29 zeigt ein Diagramm der Abstrahlcharakteristik. Das Maximum der Intensität der abgegebenen Strahlung befindet sich direkt vor der LED (0°). Mit zunehmenden Winkel nimmt die Intensität ab und erreicht bei 60° noch 50% des maximalen Wertes. Je langsamer die Intensität zu den Außenbereichen hin abfällt, desto gleichmäßiger ist die Ausleuchtung und desto besser ist eine LED für den Einsatz am tTHG geeignet . Außerdem handelt es sich um eine High-Power-LED, die mit Stromstärken von bis zu 1A betrieben werden kann und genügend Leistung liefert, um den Globenkörper auszuleuchten.

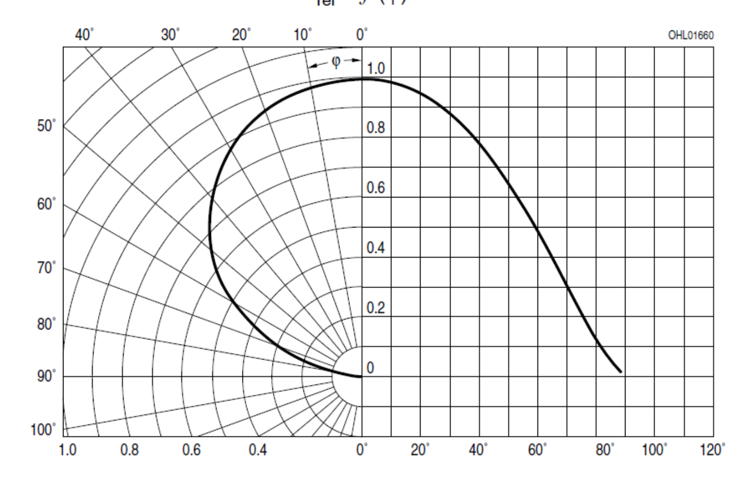


Abbildung 29: Abstrahlcharakteristik einer IR-LED, Typ Osram SFH 4235 (Quelle: [OSR-12]:4)

3) Als Chipfläche wird der Bereich einer LED bezeichnet, der aktiv Licht aussendet.

5.3.1.2 Kamera

In modernen Photo- und Videokameras werden zur Bildgewinnung CCD- und CMOS-Sensoren verbaut. Diese Sensoren reagieren sowohl auf Strahlung im Bereich des sichtbaren Lichts als auch auf Licht im unteren Bereich des nahen Infrarotspektrum, der sich von 780 bis 1400nm erstreckt (siehe Abbildung 30). Da die zur Verfügung stehenden IR-LEDs Licht der Wellenlänge zwischen 850nm und 940nm abgeben, ist eine Empfindlichkeit des Sensors nicht im ganzen nahen Infrarot-Bereich notwendig. Diese Sensitivität der Kameras für IR-Strahlung ist in der Regel nicht erwünscht, da dadurch Bilder erzeugt werden, die nicht der menschlichen Wahrnehmung entsprechen.

Daher werden spezielle Filter vor den Sensoren bzw. an den Objektiven angebracht, welche nur Licht in dem vom Menschen wahrnehmbaren Spektrum passieren lassen. Werden diese Filter durch sogenannte IR-Passfilter, die lediglich Licht im nahen Infrarotbereich passieren lassen, ersetzt, erhält man eine IR-Kamera. Eine solche modifizierte Kamera ist auch für die Erfassung von Berührungen am THG nötig.

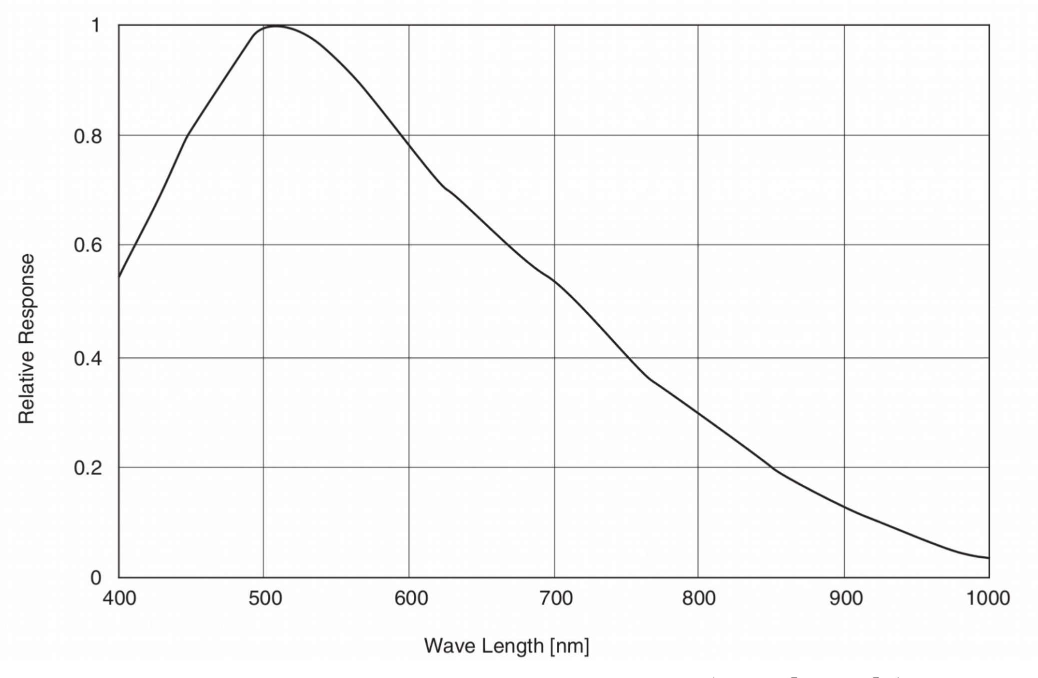


Abbildung 30: Spektrale Empfindlichkeit eines CCD-Sensors, Typ Sony ICX204AL (Quelle: [SON-12]:9)

Für die Auswahl einer geeigneten Kamera ist der Sensortypus somit unerheblich. Jedoch muss gewährleistet sein, dass sowohl am Sensor als auch an den Objektiven keine IR-Sperrfilter oder Beschichtungen verbaut sind oder sich entfernen lassen. Darüber hinaus sind die Abmessungen einer Kamera von Bedeutung. Eine zu große Kamera könnte nicht im Gehäuse des Globus installiert werden ohne die Projektion zu beeinträchtigen. Des Weiteren sollte eine Programmierschnittstelle (API) vorhanden sein, um möglichst einfach und schnell auf die Kamerabilder zugreifen zu können. Zwar ist ein Zugriff auch über die Ebene des Betriebssystems möglich (z.B. Windows DirectShow), allerdings decken diese Schnittstellen in der Regel nicht den vollen Funktionsumfang einer Kamera ab.

Eine Kamera, die alle diese Bedingungen erfüllt, ist das Modell Firei-XGA Pro der Firma Unibrain (siehe Abbildung 31). Sie liefert Bilder mit einer maximalen Auflösung von 1024 x 768 Pixel und einer Frequenz von 30 Bildern/Sekunde. Der CCD-Sensor ist auf einer eigenen Platine untergebracht und über ein Flachbandkabel mit einer weiteren Platine verbunden, auf der sich die eigentliche Steuerungselektronik befindet. Durch diese Anordnung lässt sich der begrenzte Platz innerhalb des Globenghäuses optimal ausnutzen. Zusätzlich wird vom Hersteller eine C++ API in Form einer Softwarebibliothek mitgeliefert.

Als Zubehör zur Kamera wurden eine passendes Zoomobjektiv (6 – 60 mm Brennweite) sowie ein IR-Passfilter verwendet.

Abbildung 31: Produktfoto Unibrain Fire-i XGA Pro (Quelle: [UNI-12])

5.3.2 SOFTWARE

5.3.2.1 Gesten

> „**A gesture is a motion of the body that contains information.** *Waving good-bye is a gesture. Pressing a keyboard is not a gesture because the motion of a finger on its way to hitting a key is neither observed nor significant. All that matters is which key was pressed.*“
> [WEB-12]:168 nach HULTEEN und KURTENBACH 1990, Hervorh. d. Verf.

In der Literatur über Mensch-Computer-Interaktion ist diese eine weit verbreitete Definition für Gesten. Aus ihr geht hervor, dass sich eine Geste durch die Tatsache, dass sie eine Information beinhaltet oder vermittelt, von einer reinen Bewegung unterscheidet. Diese Information ist gewissermaßen durch die Bewegung codiert. Soll eine Geste zur Steuerung eines Computersystems eingesetzt werden, muss diese codierte Information von einer Software wieder entschlüsselt werden. Dies geschieht durch die Auswertung bestimmter Eigenschaften einer Geste. SAFFER bezeichnet diese als Attribute und ordnet in [SAF-08] einer Geste folgende Eigenschaften zu:

- Präsenz
- Dauer
- Position
- Bewegung
- Druck
- Größe
- Orientierung
- betroffene bzw. einbezogene Objekte
- Anzahl der Berührungspunkte und Kombinationen
- Sequenz
- Anzahl der Teilnehmer bzw. Nutzer

Wird nun ein auf Gesten basiertes System bzw. Interface entworfen, sollte darauf geachtet werden, dass die verwendeten Gesten einige Bedingungen erfüllen. Sie sollten so gestaltet sein, dass ein Nutzer sie als möglichst natürlich und intuitiv wahrnimmt. Zusätzlich muss die Ausprägung der Attribute so beschaffen sein, dass sie durch die Steuerungssoftware eindeutig ausgewertet werden können. Weitere Hinweise und Vorschläge zur Gestaltung solcher Interfaces finden sich ebenfalls bei [SAF-08].

Wie im Konzept für die Umsetzung eines tTHG erwähnt, sollen zwei verschiedene durch Gesten ausgelöste Funktionen implementiert werden: Eine einfache Berührung und Bewegung zur Rotation des Globus sowie eine Multitouch-Geste, die im Folgenden als „einfache Rotation" und „Switch" bezeichnet werden. Bei einer einfachen Rotation soll ein bestimmter Punkt von seinem Ausgangsort an einen Zielort verschoben werden. Dazu berührt ein Nutzer diesen Punkt und durch eine Bewegung zum beabsichtigten Zielort wird der Globus rotiert. Diese Art der Steuerung soll vor allem kleinere und präzise Rotationen ermöglichen. Um eine bestimmte Region schnell ins Blickfeld zu versetzen, wird hingegen die Switch-Geste verwendet. Mit ihr soll es möglich sein zwischen den Kontinenten zu wechseln. Durch Auflegen der flachen Hand und einer weiteren Berührung des Globus wird die Rotation des Globus so verändert, dass ein Kontinent an die Position der ersten Berührung rotiert wird. Durch eine weitere Berührung wird zum nächsten Kontinent gewechselt.

Dafür müssen bei der Auswertung der Gesten folgende Attribute berücksichtigt werden: Präsenz, Position, Dauer und Bewegung sowie die Anzahl der Berührungspunkte. Diese Eigenschaften müssen aus den Bilddaten der IR-Kamera ermittelt, in Steuerungsbefehle übersetzt und an die RE übergeben werden. Im nächsten Unterkapitel werden die Techniken und Prinzipien erläutert, die für eine solche Interpretation des Bildmaterials durch eine Software notwendig sind.

5.3.2.2 Computer Vision

> *„Computer vision is the transformation of data from a still or video camera into either a decision or a new representation. All such transformations are done for achieving some particular goal.“*
> [BRA-08]:2

Computer Vision bzw. maschinelles Sehen ist ein Teilbereich der Informatik, der sich damit befasst, Informationen aus Bilddaten mit Hilfe von computergestützten Analysen zu extrahieren. Aus diesen Informationen werden Objekte identifiziert, unterschieden und ggf. klassifiziert. Aus der Art und Verteilung der Objekte können dann bestimmte Aktionen abgeleitet werden.

Applikationen die maschinelles Sehen einsetzen sind weit verbreitet. Sie werden in der Verkehrsüberwachung benutzt, um Zählungen durchzuführen oder Staus und Unfälle zu erkennen. Sie eignen sich aber auch zur Überwachung von Produktionslinien in der industriellen Fertigung oder zur Erkennung und Identifikation von Gesichtern und Personen.

Diese angeführten Beispiele sind allerdings ungleich komplexer und aufwendiger als die Erkennung von Gesten am tTHG. Die Bilder, die von der IR-Kamera dazu geliefert werden, sind im Idealfall vollständig schwarz, d.h. es dringt kein IR-Licht in die Kamera ein und alle Pixel weisen den Helligkeitswert 0 auf[4]. Findet eine Berührung statt, wird wie in Abbilung 27 dargestellt ein Teil des IR-Lichts zur Kamera hin reflektiert. Diese werden durch helle Bereiche im Bild bzw. Pixel mit einem Farbwert, größer als 0, dargestellt. Das Eintreten dieses idealen Falles ist bei der Umsetzung des tTHG jedoch nicht zu erwarten.

Durch Einflüsse von außen (wie z.B. Tageslicht oder andere künstliche Lichtquellen) und die relativ geringe Empfindlichkeit des CCD-Sensors im Bereich des eingesetzten IR-Lichts werden Störungen im Bild verursacht, die durch eine Vorverarbeitung weitestgehend wieder entfernt werden sollen. Mindestens zwei Schritte sind dazu nötig.Als Erstes wird durch die Anwendung eines Weichzeichnungsfilter das Bild geglättet. Somit lassen sich kleine, ungewollte Reflexionen und Rauschen beseitigen. Im zweiten Schritt wird die angesprochene geringe Sensitivität im IR-Bereich durch eine Histogrammspreizung ausgeglichen. Bei diesem Verfahren werden zunächst der minimale und maximale Helligkeitswert im Bild bestimmt. Dieser Wertebereich wird linear auf den maximalen Bereich (0 – 255) gespreizt und den Pixeln die daraus resultierenden Werte zugewiesen. Dieses Verfahren ist für die eigentliche Bil-

4) Die IR-Kamera gibt monochromatische Bilder mit einer Farbtiefe von 8 Bit aus. Daher weisen die einzelnen Pixel Werte zwischen 0 und 255 auf (entspricht 256 Graustufen).

derkennung nicht unbedingt notwendig, erleichtert aber die visuelle Überprüfung der Ergebnisse. Im letzten Schritt der Vorverarbeitung werden die Bilder in ein binäres Format übertragen. Den Pixeln wird je nachdem ob ihr Helligkeitswert größer oder kleiner als ein bestimmter Schwellenwert ist der neue Wert 1 bzw. 0 zugewiesen.

Anschließend werden durch einen Algorithmus zusammenhängende Bereiche, die den Wert 1 aufweisen, erkannt. Diese Bereiche werden auch als Blobs bezeichnet. Neben einer ID zur Identifikation werden Informationen wie die Bildkoordinaten des Zentrums, der Umriss und die Fläche des Blobs ermittelt. Lassen sich Blobs, die Ähnlichkeiten in ihrer Position und Größe aufweisen, in mehreren aufeinanderfolgenden Bilder beobachten, werden diese zu sogenannten Tracks zusammengefasst. Einem solchen Track-Objekt wird neben einer eigenen ID die ID des aktuellen, dazugehörigen Blobs, das Alter (Anzahl der Bilder, in denen der Track existiert) sowie Angaben über die Bewegungsrichtung zugewiesen. Aus diesen Blobs und Tracks lassen sich nun Gesten ableiten und entsprechende Reaktionen formulieren.

5.3.2.3 Freie Bibliotheken

Die Algorithmen, die für die oben beschriebene Bildverarbeitung benötigt werden, sind zum Teil sehr komplex und aufwendig. Eine eigene, zeitaufwendige Umsetzung der dahinterstehenden mathematischen Modelle in lauffähigen Quellcode ist jedoch nicht Ziel dieser Arbeit und darüber hinaus besteht keine Notwendigkeit, da sie in Form von freien Software-Bibliotheken bereits zur Verfügung stehen. Bei diesen Bibliotheken handelt es sich in diesem Fall um eine Sammlung von in Funktionen implementierten Algorithmen, die über ein API von anderen Programmen benutzt werden können. Sie ermöglichen es modulare Software zu entwickeln, deren einzelne Teile in andere Applikationen wiederverwendet werden können.

Sämtliche für den tTHG notwendigen Bildverarbeitungsroutinen finden sich in der OpenCV (CV steht für Computer Vision) Bibliothek (siehe [OPC-12]). Diese ursprünglich von der Firma Intel entwickelte Bibliothek ist quelloffen und für nicht kommerzielle und akademische Anwendungen frei verfügbar. Darüber hinaus ist diese Bibliothek bei Entwicklern und Bastlern sehr beliebt, sodass sich viele Foren und Tutorials finden lassen, die Hilfestellung bei Problemen geben können. Zwar verfügt OpenCV neben den Funktionen, die die Bild(vor)verarbeitung betreffen auch über Routinen zur Erkennung von Blobs und Tracks. Jedoch wurde für die Umsetzung der tTHG-Software auf eine andere, ebenfalls frei verfügbare Bibliothek, mit der Bezeichnung cvblob (siehe [CVB-12]) zurückgegriffen. Der Algorithmus, der in dieser Bi-

bliothek angewandt wird, ist zwar weniger ausgereift und wird nur für Tests und Prototypen empfohlen, besitzt aber den Vorteil, dass diese sich wesentlich einfacher implementieren lässt als die OpenCV-Variante.

5.4 REALISIERUNG

5.4.1 HARDWAREMODIFIKATION

Die IR-LEDs SFH4235 (siehe Abbildung 32) werden in einer SMD-Bauform produziert. SMD steht für „Surface-mounted device“ (deutsch: auf einer Oberfläche montiertes Bauteil). Diese, in der Elektronik mittlerweile übliche Bauweise, ermöglicht es Bauteile zu miniaturisieren und sehr dicht auf einer Platine zu platzieren. Die manuelle Verarbeitung wird dadurch allerdings erschwert. Es erfordert einige Übung die stromleitenden Kabel an den Anschlußstellen, die nur 1,8 x 1,0 mm groß sind, anzulöten. Die empfohlene Betriebstemperatur darf laut Datenblatt des Herstellers 125 °C nicht überschreiten (vgl. [OSR-12]).

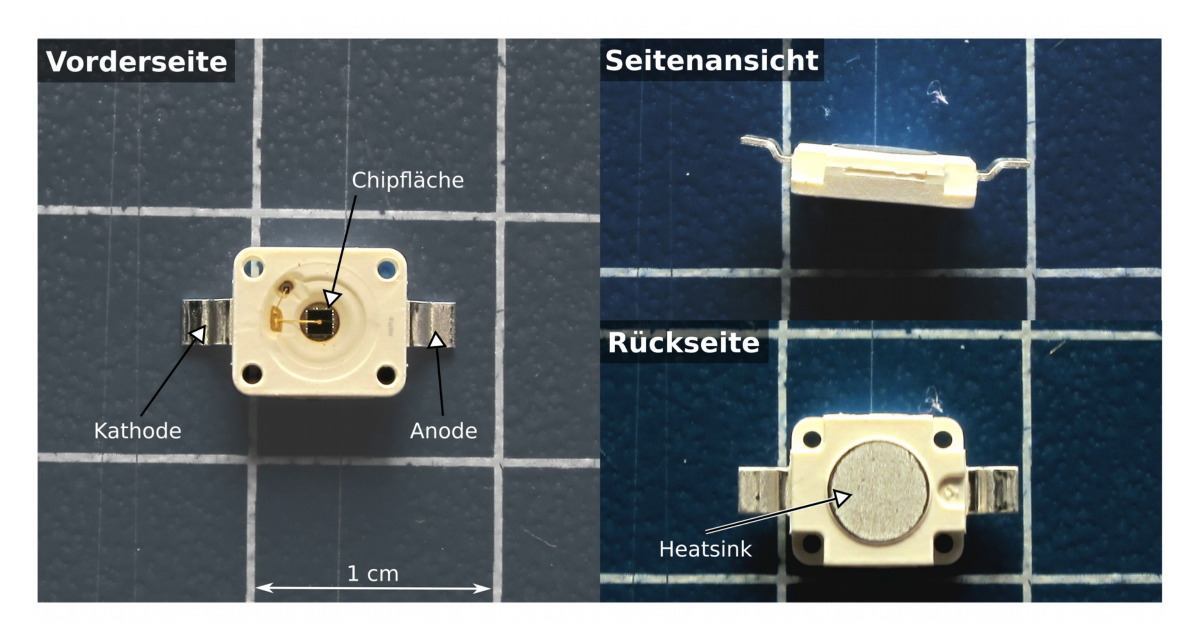


Abbildung 32: IR-LED, Typ Osram SFH4235, Detailaufnahme (Quelle: eigene Aufnahme)

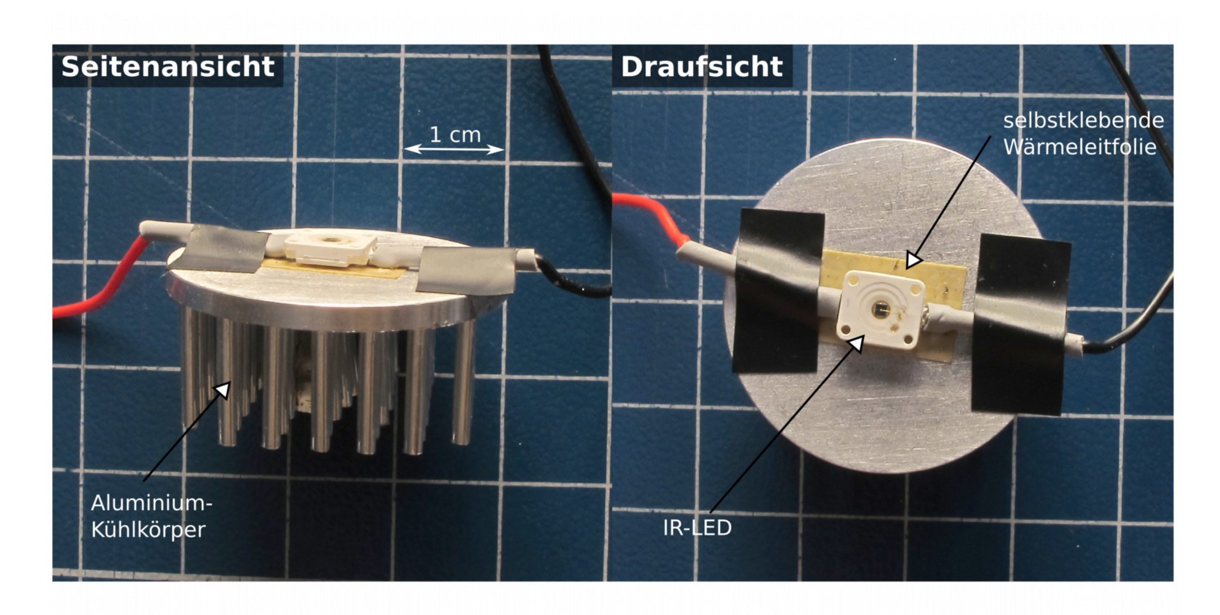


Abbildung 33: IR-LED, Typ Osram SFH4235, montiert auf Kühlkörper (Quelle: eigene Aufnahme)

Um dies zu gewährleisten, wurden die LEDs auf Kühlkörper aus Aluminium angebracht (siehe Abbildung 33). Dazu wurde eine doppelseitig selbstklebende Wärmeleitfolie verwendet. Diese stellt eine Fixierung der LED sicher und gewährleistet einen raschen Abtransport der Wärme von der LED zum Kühlkörper. Insgesamt wurden fünf Kühlkörper mit je einer LED bestückt. Die Befestigung dieser Einheiten an der Ghost-trap erfolgte mittels Magneten. Diese flexible Art der Befestigung erleichtert die Bestimmung einer möglichst optimalen Position. Am günstigsten erwies sich dabei eine Montage am oberen Rand der Ghost-trap. Die fünf Einheiten wurden in Reihe zu einem Stromkreis verbunden. Die dazu nötigen Kabel wurden ebenfalls an der Ghost-trap fixiert und die Zuleitungen wurden innerhalb dieser verlegt und aus dem Gehäuse nach außen geführt. Die Stromversorgung erfolgt über ein Labornetzteil. Durch eine am Netzteil einstellbare Stromstärkenbegrenzung wurde sichergestellt, dass es zu keiner Überbelastung der LEDs kommen kann. Abbildung 34 zeigt die fertig modifizierte Ghost-trap.

Abbildung 34: Ghost-trap mit befestigten IR-LEDs und Verkabelung (Quelle: eigene Aufnahme)

Wie in Kapitel 5.3.1.2 beschrieben, besteht die IR-Kamera aus zwei getrennten Platinen. Die Regelungselektronik und der FireWire Anschluß befinden sich auf der größeren Platine, die Abmessungen von ca. 4,5x7 cm besitzt. Die 3x4 cm messende kleinere Platine beherbergt den CCD-Sensor und einen Adapter für CS- bzw. C-Mount kompatible Objektive (siehe Abbildung 35). In diesen Adapter wird ein IR-Passfilter eingesetzt, um nur Licht der gewünschten Wellenlänge passieren zu lassen. Das verwendete Zoomobjektiv besitzt manuelle Regler zum Anpassen der Brennweite und des Fokus. Außerdem verfügt es über eine elektronisch regelbare Steuerung zum Öffnen und Schließen der Iris. Für die Anwendung am tTHG ist diese aber nicht nötig bzw. eine vollständig geöffnete Iris erwünscht, um so viel Licht wie möglich auf den CCD-Sensor weiterzuleiten. Daher wird an den Anschlüssen für diese Regelung die dafür erforderliche konstante Spannung von 3 V angelegt.

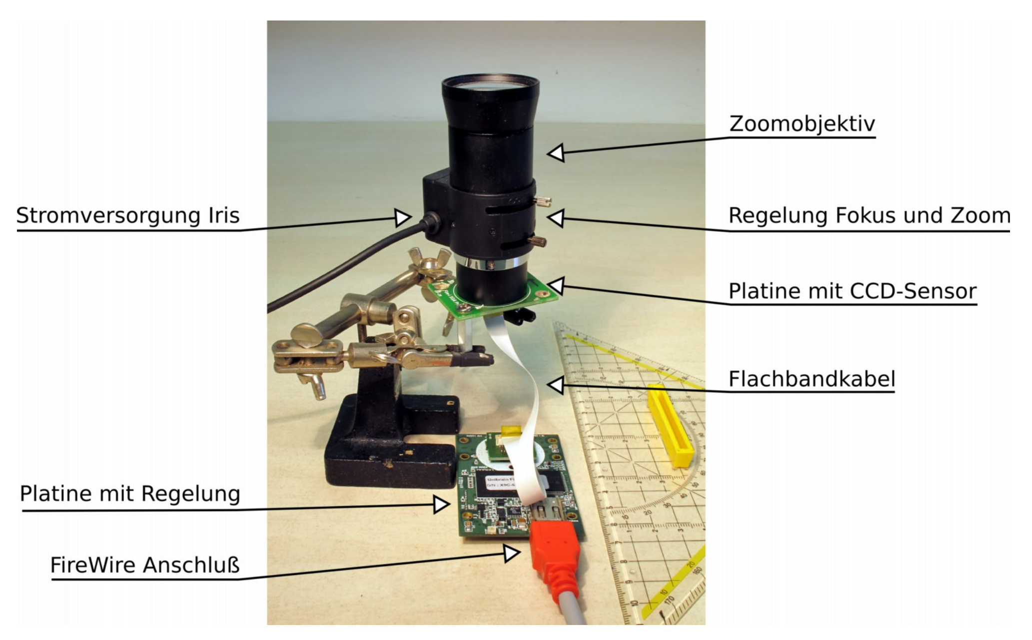


Abbildung 35: IR-Kamera, Objektiv und Befestigung (Quelle: eigene Aufnahme)

Abbildung 36: IR-Kamera, Positionierung im Globus (Quelle: eigene Aufnahme)

Die Positionierung der Kamera im Gehäuse des Globus zeigt die Abbildung 36. Die CCD-Platine wird mit einer sogenannten dritten Hand fixiert und so nahe wie möglich unter dem Mittelpunkt des Disperser platziert. Das Objektiv sollte so ausgerichtet sein, dass das Zentrum des Dispersers (= Projektionszentrum) in der Bildmitte abgebildet wird. Die Brennweite ist so zu wählen, dass der Disperser das Bild vollständig ausfüllt.

5.4.2 SOFTWARE

Im Folgenden wird der Aufbau und die Abläufe in der für den tTHG entwickelten Software erläutert. Um die Lesbarkeit und die Verständlichkeit zu erhöhen, wird zur Erläuterung der Funktionen auf `Pseudocode` zurückgegriffen. Der vollständige C++ Quellcode findet sich im Anhang zu dieser Arbeit.

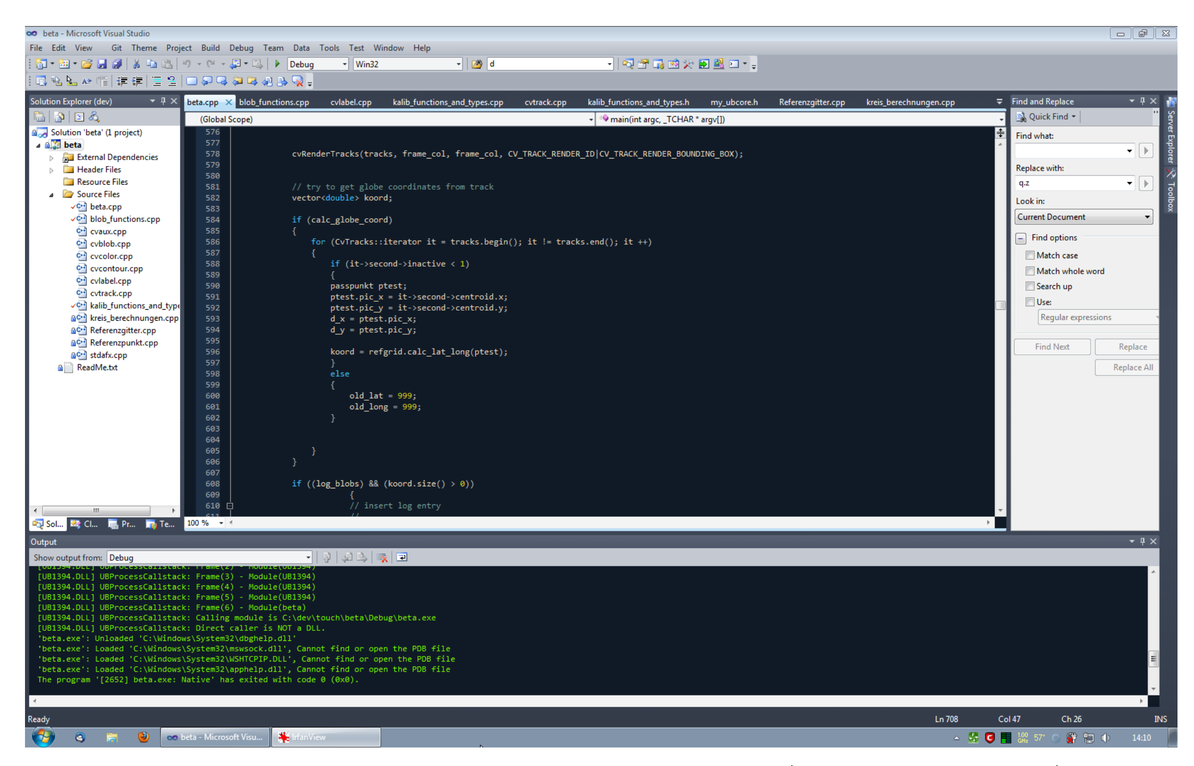


Abbildung 37: Screenshot der Entwicklungsumgebung MS Visual Studio 10 (Quelle: eigene Darstellung)

Der Vorgang der Auswertung der in den Kamerabildern enthaltenen Gesten wird in kleinere Aufgabenpakete zerlegt und in Funktionen umgesetzt. Um während der Entwicklung des Programms die Ergebnisse jederzeit visuell kontrollieren zu können, wird als Erstes eine GUI entworfen. Diese besteht aus drei Fenstern, in denen das unveränderte Bild der Kamera, eine gefilterte Version dieses Bildes und die erkannten Blobs bzw. Tracks angezeigt werden können. Zusätzlich besteht die Möglichkeit über eine Konsole Text auszugeben und über verschiedene Regler und Tastatureingaben das Programm zu beeinflussen.

Der nächste Schritt ist die Herstellung einer Verbindung zur IR-Kamera und das Entwerfen einer Funktion, die es erlaubt Bilder von der Kamera zu übertragen und in ein zu OpenCV kompatibles Format umzuwandeln. Diese Rohdaten werden nun durch die Anwendung verschiedener Algorithmen der Bildbearbeitung aufbereitet und in ein binäres Format umge-

wandelt, sodass den Pixeln nur mehr die Werte 0 oder 1 zugeordnet sind. Anschließend erfolgt die Detektion der Blobs und Tracks mit Hilfe der durch die cvblob-Bibliothek zur Verfügung gestellten Funktionen. So erhält man die Orte der Berührungen in zweidimensionalen Bildkoordinaten. Für die weiteren Berechnungen ist es jedoch notwendig diese in Kugelkoordinaten bzw. geographische Koordinaten (Länge und Breite) übertragen zu können. Um diese Transformation durchführen zu können, muss das Kamerabild kalibriert werden. Bei diesem Vorgang werden vorgegebene Punkte am Globus berührt und die dazugehörigen Bildkoordinaten gemessen. Mit Hilfe dieser Referenzwerte kann nun zu allen Bildkoordinaten die Lage am Globus bestimmt werden. Für die Unterscheidung zwischen einer Geste zur einfachen Rotation und der Switch-Geste ist es notwendig einzelne Tracks in stationäre und dynamische Berührungen zu unterteilen. Dies erfolgt durch die Auswertung des Verhältnisses von Lebensdauer zur Entfernung zwischen dem aktuellen und den initialen Zentrum der jeweiligen Tracks.

Der Ablauf des Programms lässt sich in zwei Abschnitte gliedern (siehe Abbildung 38). In einer ersten Phase werden alle notwendigen Komponenten initialisiert. Dies betrifft die GUI, die Verbindung zur Kamera, die Netzwerkschnittstelle zur Übertragung von Befehlen an die RE und eine Vielzahl von Variablen, die für die weiteren Schritte benötigt werden. Auch die Kalibrierung der Kamera erfolgt in diesem Abschnitt. Anschließend wird eine Schleife gestartet, die solange wiederholt wird, bis der Benutzer den Befehl zum Abbrechen und Beenden des Programms übermittelt. In dieser Schleife wird jeweils ein Bild von der Kamera übertragen. Nach der Aufbereitung des Bildes werden die Blobs und Tracks bestimmt, die Gesten ausgewertet und entsprechende Kommandos an die RE übergeben. Am Ende der Schleife werden eventuelle Nutzereingaben registriert und ausgewertet.

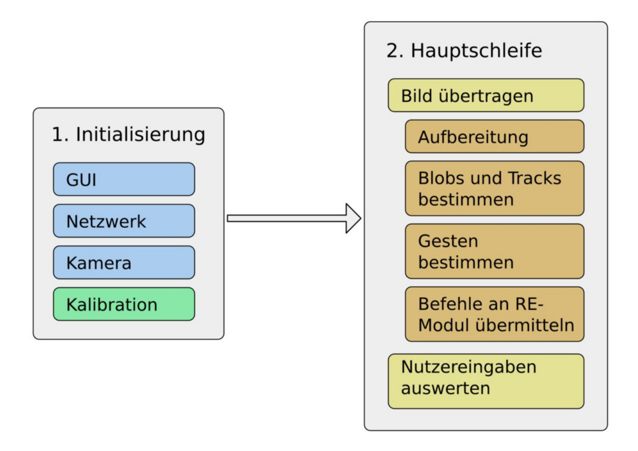


Abbildung 38: tTHG Software, Programmablauf (Quelle: eigene Darstellung)

Diese Beschreibung der Funktionen und Abläufe innerhalb der Software soll nur einen Überblick liefern. Die noch fehlenden Details werden in den folgenden Abschnitten besprochen.

5.4.2.1 Verbindung zur Kamera

Der Quelltext für die Initialisierung der IR-Kamera wurde aus einem vom Hersteller zur Verfügung gestellten Beispiel entnommen. Diese Beispiele und die Dokumentation der API sind auf dessen Website zu finden (siehe [UNI-12a]). Die einzelnen Schritte die während dieser Prozedur ausgeführt werden sind im Folgenden beschrieben:

- `FiInitialize():`
  Initialisieren der API.

- `FiLocateCameras():`
  Diese Funktion sucht nach mit dem Rechner verbundenen Kameras und liefert für jede gefundene Kamera eine eindeutige ID als Rückgabewert.

- `FiOpenCameraHandle():`
  Hier wird ein Handle-Objekt erstellt. Über dieses Objekt erfolgt die Steuerung der Kamera.

- `FiStopCamera():`
  Um sicherzugehen, dass die Kamera nicht bereits mit einem Programm verbunden ist (z.B. nach einem Absturz des Programms), wird an dieser Stelle versucht alle aktiven Verbindungen zu trennen und die Kamera in einen nichtaktiven Modus zu versetzen.

- `FiStartCamera():`
  Dieser Funktion werden die Verbindungsparameter übergeben. Diese beinhalten Angaben zur gewünschten Bildauflösung, Farbtiefe, Bildwiederholrate und Übertragungsgeschwindigkeit. Gemäß dieser Parameter wird die Kamera anschließend in einen aktiven Modus versetzt.

- `FiCreateIsochReceiveEngine():`
  Dieser Aufruf beendet die Initialisierung. Dabei wird auf ein Objekt vom Typ `IsochRecieveEngine` verwiesen. Über dieses Recieve-Objekt erfolgt die eigentliche Übertragung der Bilddaten von der Kamera an die Software.

Diese Übertragung wird durch die Funktion `get_frame()` realisiert. Ihr werden als Parameter Zeiger auf das Recieve-Objekt sowie auf eine Variable vom Typ `IplImage` übergeben. Innerhalb der Funktion werden die Rohdaten von der Kamera übertragen, in ein OpenCV-kompatibles Format umgewandelt und der Variablen zugewiesen.

5.4.2.2 Bildvorverarbeitung, -aufbereitung

Die Aufbereitung der Bilder erfolgt durch die in der OpenCV-Bibliothek verfügbaren Funktionen zur Bildbearbeitung. Die Funktion dieser Filter wurde bereits im Abschnitt über Computer Vision kurz erläutert. Die vollständige Dokumentation aller OpenCV-Funktionen findet sich auf der Website der Bibliothek (siehe [OPC-12a]). Diese Filter wurden im Programm so implementiert, dass ihre Parameter über Schieberegler oder Tastatureingaben während des Betriebs verändert werden können.

- `CvThreshold():`

  Diese Funktion wird benutzt, um ein Bild zu binarisieren. Gemäß eines als Parameter an die Funktion übergebenen Schwellenwerts, werden allen Pixeln eines Bildes die Werte 0 oder 1 zugewiesen, je nach dem ob ihr Ursprungswert kleiner oder größer als der Schwellenwert ist.

- `CvEqualizeHist():`
  Eine Funktion, die aufgerufen wird, um eine Histogrammspreizung durchzuführen.

- `CvSmooth():`
  Um kleinräumige Störungen und Rauschen im Bild zu entfernen, wird ein Gaußscher Weichzeichnungsfilter angewandt. Dieser Filter weist jedem Pixel den Durchschnittswert der ihn umgebenden Pixel zu. Über einen Parameter lässt sich der Radius, der als Umgebung betrachtet wird, bestimmen.

Für die Erkennung der Blobs und Tracks wird nur die Funktion `CvSmooth()` angewendet. Die Histogrammspreizung und Binarisierung erfolgen innerhalb der Funktionen, die durch die cvblob-Bibliothek zur Verfügung gestellt werden. Der Grund für die oben beschrieben Implementierung ist, dass sie für die visuelle Interpretation und Fehlersuche hilfreich sind.

5.4.2.3 Kalibration

Die Kalibration der Kamera umfasst zwei Schritte. Zum einen die Beseitigung von Störungen, die durch Fremdlicht und Reflexionen verursacht werden, und zum anderen die Erstellung einer Abbildungsvorschrift zur Transformation der zweidimensionalen Bildkoordinaten in geographische Koordinaten am Globus.

Im Laufe der Tests zur Positionierung der LEDs hat sich ergeben, dass die Innenseite des Globus die IR-Strahlung nicht gleichmäßig und vollständig passieren lässt. Dies führt zu dauerhaften Reflexionen, die auch im Bild der Kamera zu erkennen sind. An diesen Stellen ist eine Detektion von Berührungen nicht möglich. Mit Hilfe der Funktion `blind_spot_kalib()` werden diese Bereiche identifiziert und eine Maske generiert (siehe Abbildung 44). Sie wird unmittelbar nach der Initialisierung der Kamera aufgerufen.

```
blind_spot_kalib()

	// 1. Gesammelte maximale Pixelwerte über 60 Bilder ermitteln
	Wiederhole 60 mal:
		get_frame()
		Histogrammspreizung durchführen
		Maximalwerte in ein temporäres Bild BT übertragen

	// 2. BT binarisieren und Werteverteilung ermitteln
	für x größer gleich 0 bis x kleiner 255:
		BT mit Schwellenwert x binarisieren
		Häufigkeit des Werts 1 ermitteln und abspeichern

	// 3. Größte Differenz zwischen zwei Häufigkeiten
	D = 0
	für x größer gleich 0 bis x kleiner 254:
		Differenz d  = Werteverteilung(x) - Werteverteilung(x+1)
		Wenn d größer als D:
			D = x+1

	// 4. Maske erstellen
	BT mit dem Schwellenwert D+1 binarisieren
	Weichzeichnungsfilter auf BT anwenden
	BT zurückgeben
```

Um den Schwellenwert, ab dem ein Pixel als zu hell und somit als unbrauchbar für die Detektion von Berührungen angesehen wird, herauszufinden, sucht diese Funktion nach dem größten Sprung in der Werteverteilung aller Pixel. Durch Tests konnte bestätigt werden, dass diese Methode, zumindest für den am Institut vorhandenen Globus, brauchbare Ergebnisse erzielt. Diese so erstellte Maske wird im weiteren Verlauf des Programms von jedem Kamerabild abgezogen, sodass die nicht tauglichen Bereiche den Wert 0 aufweisen.

Der zweite Schritt der Kalibration dient dazu, eine Abbildungsvorschrift zwischen den Pixel-

koordinaten eines Blobs oder Tracks im Kamerabild und den geographischen Koordinaten einer Berührung am Globus zu erstellen. Dazu sollen als erstes die Zusammenhänge zwischen diesen beiden Koordinatensystemen erläutert werden (siehe Abbildung 39). Ein Breitenkreis am Globus wird als Kreis mit dem Mittelpunkt des Dispersers als Zentrum abgebildet. Meridiane erscheinen als gerade Linien, die sich vom Zentrum des Dispersers bis zu dessen Rand erstrecken. In einem polaren Koordinatensystem, dessen Ursprung im Mittelpunkt des Disperser liegt und dessen Punkte die Koordinaten [r,Φ] aufweisen, lassen sich die geographische Breite θ als Funktion des Radius r und die geographische Länge φ als Funktion des Winkels Φ ausdrücken.

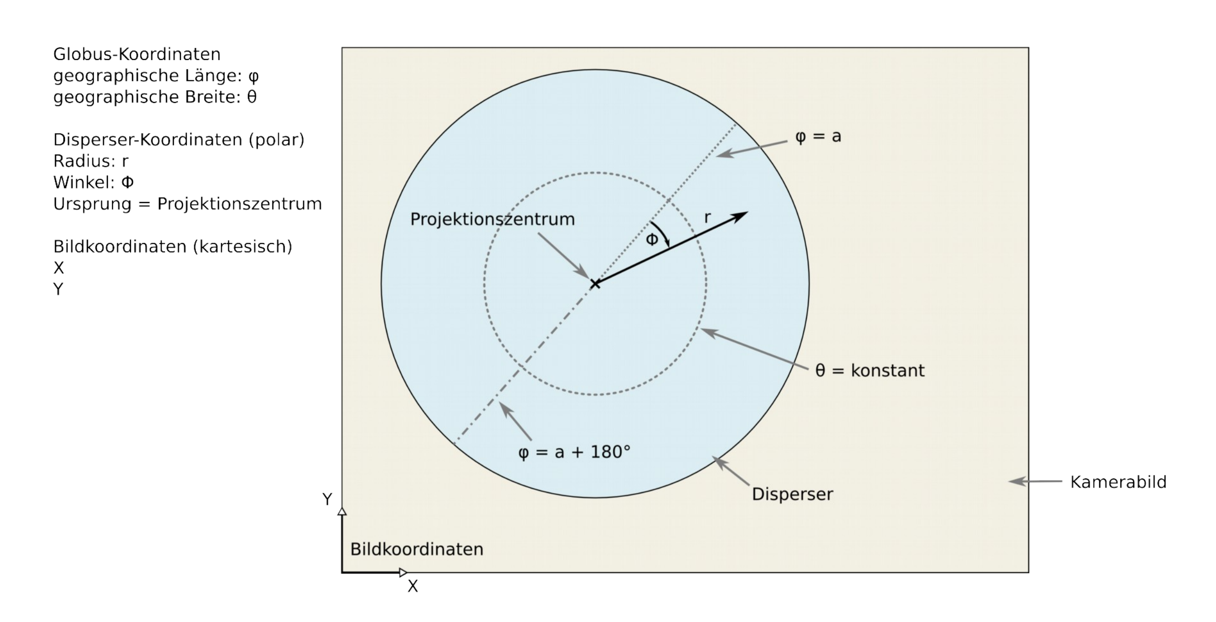


Abbildung 39: verschiedene Koordinatensysteme im Kamerabild (Quelle: eigene Darstellung)

Der Vorgang der Zuordnung eines Pixels im Bild zu einem Punkt am Globus erfordert also die Transformation der X-Y-Bildkoordinaten in das oben erwähnte Polarkoordinatensystem und anschließend die Überführung in eine geographische Länge und Breite. Zunächst wird der Ursprung des Polarkoordinatensystems bestimmt. Dazu wird am Globus ein regelmäßiges Gitternetz mit 20° Abständen eingeblendet. Nun wird jeweils einer der Gitterpunkte berührt und die dazugehörigen geographischen Koordinaten per Tastatureingabe übermittelt. Dadurch entsteht eine Liste von Referenzpunkten, deren geographische Koordinaten als auch ihre Bildkoordinaten bekannt sind. Aus dieser Liste werden alle Kombinationen $K(a,b,c)$ aus drei Elementen mit gleicher geographischer Breite gebildet. Für jede dieser Kombinationen gilt, dass ihre drei Elemente Punkte eines Kreises sind und das Zentrum dieses Kreises der Schnittpunkt der Mittelsenkrechten der Strecken $\overline{ab}$ und $\overline{bc}$ ist. Um Ungenauigkeiten

bei der Erfassung der Referenzpunkte auszugleichen, werden diese errechneten Kreismittelpunkte gemittelt und der daraus resultierende Punkt als Ursprung dem Polarkoordinatensystem zugewiesen. Abschließend werden für jeden Referenzpunkt die Polarkoordinaten an Hand der Formeln

$r=\sqrt{x^2+y^2}$ und $\Phi=atan2(y,x)$

berechnet. Für die Bestimmung eines unbekannten Punktes werden an Hand der Polarkoordinaten die vier ihn umgebenden Referenzpunkte bestimmt und die geographischen Koordinaten durch Interpolation berechnet. Sofern zwischen zwei Programmabläufen die Position der Kamera nicht verändert wurde, können diese Referenzpunkte wiederverwendet werden. Daher wurde eine Funktion implementiert, die das Abspeichern und Laden von Referenzpunkten in eine bzw. aus einer Textdatei ermöglicht.

5.4.2.4 Blob-, Track- und Gestenerkennung

Um aus den Kamerabildern die Gesten zu bestimmen, müssen zunächst die Blobs und Tracks erkannt werden. Dies geschieht über den Aufruf zweier Funktionen aus der cvblob-Bibliothek. Diese Funktionen liefern jeweils eine Liste mit allen erkannten Blob- und Track-Objekten zurück. Dabei ist zu beachten, dass die Liste der Blobs für jedes analysierte Bild neu erstellt wird. Die Liste der Tracks hingegen wird für jeden Frame lediglich aktualisiert. Anhand festgelegter zeitlicher Grenzwerte wird ein Track aus der Liste entfernt, wenn keiner der Blobs diesem Track zugeordnet werden kann. Für die Unterscheidung der beiden geplanten Gesten ist zudem eine Einteilung der Tracks in statische und dynamische Tracks notwendig. Ein Track bzw. eine Berührung wird dann als statisch bezeichnet, wenn die Entfernung zwischen dem aktuellen Punkt der Berührung und dem Zentrum der ersten Berührung kleiner als ein festgelegter Grenzwert ist. Dieser Grenzwert verhindert die fälschliche Klassifikation als dynamischer Track bei kleinen und unbeabsichtigten Bewegungen.

Nach dieser Einteilung können die Gesten ausgewertet werden. Für den Fall, dass lediglich eine Berührung erfasst und diese als dynamisch klassifiziert wurde, wird diese als Geste zur einfachen Rotation interpretiert und ein Befehl zur dementsprechenden Rotation des Globus an die RE übermittelt. Zwei erkannte Tracks werden als Switch-Geste ausgelegt und der Globus so rotiert, dass der nächste Kontinent zur Position des statischen Tracks rotiert wird.

```
Solange kein Befehl zum Abbrechen erfolgt:

	// 1. Kamerabild übertragen
	get_frame()

	// 2. Blobs und Tracks bestimmten
	Erstelle Liste L_b aller Blobs im aktuellen Bild
	Aktualisiere die Liste aller Tracks L_t an Hand der Blobs in L_b
	Für alle neuen Tracks:
		Speichere die Koordinaten des Zentrums als Z

	// 3. Tracks klassifizieren
	Für alle Tracks in L_t:
		Berechne die Entfernung D zwischen dem Zentrum und Z
		Wenn D > als X:
			Markiere Track als dynamisch
		Andernfalls:
			Markiere Track als statisch

	// 4. Gesten bestimmen
	A = Anzahl der aktuellen Tracks ( = Berührungen)
	Wenn A = 1 und Track = dynamisch:
		Geste als einfache Rotation erkannt
		Wenn im vorherigen Bild auch eine einfache Rotation erkannt wurde:
			Führe Rotation aus
		Aktuelle Track-Koordinaten abspeichern
	Wenn A = 2:
```

```
            Geste als Switch-Geste erkannt
            Führe Rotation zum nächsten Kontinent aus
    Wenn A = 1 und Track = statisch:
            Geste als Switch-Geste erkannt
            Warte auf weitere Berührung = Beginne von Vorne
```

5.4.2.4.1 ROTATION

Um nach dem Erkennen einer Rotations-Geste diese Rotation am Globus auszuführen, muss der RE ein entsprechendes Signal übermittelt werden. Die OS-API sieht dafür eine Textnachricht in der Form `StoryAddQuaternion [double]w [double]x [double]y [double]z` vor. Die Variablen w,x,y und z stehen dabei für die einzelnen Skalare eines Quaternion. Ein Quaternion wird wie folgt definiert:

> *Quaternionen sind verallgemeinerte komplexe Zahlen der Form*
> *w + **i**x + **j**y + **k**z,*
> *wobei w,x,y,z reelle Zahlen sind und die imaginären Einheiten [...] **i,j,k** [...]*
> [BRO-08]:294

Mit Hilfe von Quaternionen lassen sich unter anderem Rotationen und Skalierungen in einem dreidimensionalen Raum effizient beschreiben und werden daher sehr häufig im Umfeld von 3D-Darstellungen, 3D-Animationen und sonstiger 3D-Software verwendet. Eine Rotation wird dabei durch ein Rotations-Quaternion der Form $Q_{ROT} = [w, x, y, z]$ ausgedrückt. Um eine Skalierung auszuschließen, muss für den Betrag des Rotations-Quaternion folgende Bedingung erfüllt sein:

$$|Q_{ROT}| = \sqrt{w^2 + x^2 + y^2 + z^2} = 1$$

Verbildlicht lässt sich eine solche Rotation als Drehung um den Winkel

w um eine Drehachse $\vec{v}_d = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ beschreiben.

Eine entgegengesetzte Drehung wird durch die Invertierung des Quaternions beschrieben. Diese entspricht der Umkehrung der Drehachse.

$$-1 \cdot Q_{ROT}[w, x, y, z] = Q_{ROT}[w, -x, -y, -z]$$

Durch die Multiplikation zweier Quaternionen werden zwei Rotationen aufeinander folgend ausgeführt.

$$Q_1 \cdot Q_2 = \begin{bmatrix} w_1 \\ x_1 \\ y_1 \\ z_1 \end{bmatrix} \cdot \begin{bmatrix} w_2 \\ x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2 \\ w_1 x_2 + x_1 w_2 + y_1 z_2 - z_1 y_2 \\ w_1 y_2 - x_1 z_2 + y_1 w_2 + z_1 x_2 \\ w_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 w_2 \end{bmatrix}$$

Zu beachten ist, dass diese Operation nicht kommutativ ist.

$$Q_1 \cdot Q_2 \neq Q_2 \cdot Q_1$$

Für die Umsetzung in der Software bedeutet dies, dass aus den aus zwei aufeinanderfolgenden Bildern ermittelten geographischen Koordinaten der Berührungspunkte P und P' ein Quaternion abgeleitet werden muss, welches einer Rotation entspricht durch die der Punkt P nach der Rotation an den Koordinaten des Punktes P' zu liegen kommt.

```
Ausgangsrotation Rot_a = [1,0,0,0]
Berührung P = (0,0)

Schleife:
	aktuelle Berührung P' = (lat_1,lon_1)
	vorhergehende Berührung P = (lat_0, lon_0)

	// 1.
	Umrechnen der geographischen Koordinaten in x,y,z-Koordinaten der
	Einheitskugel

	Vektor v_1 = (P.x, P.y, P.z)
	Vektor v_2 = (P'.x, P'.y, P'.z)

	// 2. Bestimmung der Drehachse
	Vektor Drehachse v_d = Kreuzprodukt(v_1,v_2)

	// 3. Bestimmung des Drehwinkels = Winkel zwischen v_1 und v_2
	Winkel w = acos ( Skalarprodukt(v_1,v_2) / (Betrag v_1 * Betrag v_2))

	// 4. Überführung der Koordinaten von v_d in das lokale Koordinatensystem
	// von Rot_a
	Berechnung einer Rotationsmatrix M_Rot_a aus Rot_a
	angepasste Drehachse v_d_l = v_d * M_Rot_a

	// 5. Erstellung eines Rotations-Quaternion
	Q_rot = [w,v_d_l]

	// 6. Übemittlung an die RE
	Sende Text „StoryAddQuaternion Q_rot.w Q_rot.x Q_rot.y Q_rot.z“

	// 7. Anwendung des Rotation-Quaternion auf die Ausgangsrotation
	Q_erg = Rot_a * Q_rot

	// 8. Variablen aktualisieren
	Rot_a = Q_erg
	P = P'
```

Für die Durchführung einer reinen Rotation ist die Bestimmung eines Einheitsquaternion notwendig ($|Q|=1$). Die kartesischen, räumlichen Koordinaten x, y, z der beiden Berührungspunkte $P(\varphi_0, \lambda_0)$ und $P'(\varphi_1, \lambda_1)$ werden daher für eine Einheitskugel mit Radius $r=1$ bestimmt.

$$\rho=\frac{1}{2}\pi-\varphi \qquad \nu=\begin{Bmatrix} \lambda-2\pi & \mathit{wenn} & \lambda>\pi \\ \lambda & \mathit{wenn} & \lambda\leqslant\pi \end{Bmatrix}$$

$$x=-1\cdot\sin(\rho)\cdot\cos(\nu) \qquad y=-1\cdot\sin(\rho)\cdot\sin(\nu) \qquad z=\cos(\rho)$$

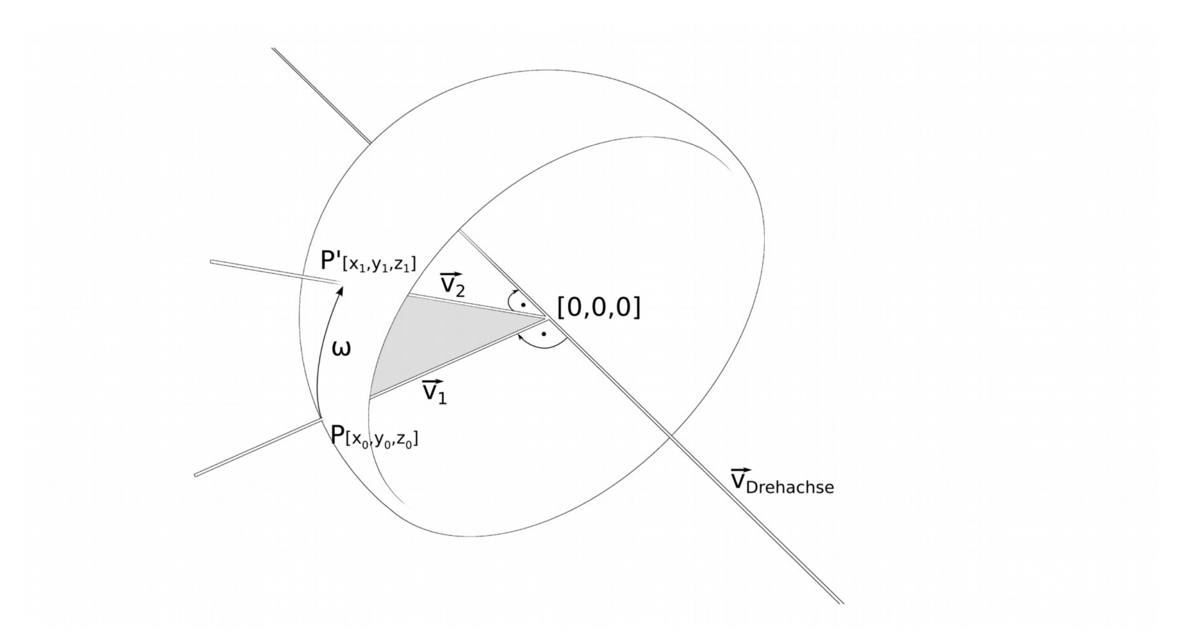


Abbildung 40: Drehung an einer Kugel (Quelle: eigene Darstellung)

In Abbildung 40 wird eine solche Situation skizziert.

Die Berechnung der Drehachse $\vec{v}_d$ erfolgt über das Kreuzprodukt der beiden Vektoren $\vec{v}_1$ und $\vec{v}_2$:

$$\vec{v}_d=\vec{v}_1\times\vec{v}_2=\begin{pmatrix}x_1\\y_1\\z_1\end{pmatrix}\times\begin{pmatrix}x_2\\y_2\\z_2\end{pmatrix}=\begin{pmatrix}y_1z_2-z_1y_2\\x_1z_2-z_1x_2\\x_1y_2-y_1x_2\end{pmatrix}$$

Der Drehwinkel ω entspricht dem Winkel den $\vec{v}_1$ und $\vec{v}_2$ bilden. Dieser wird berechnet durch:

$$\cos\omega = \frac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_1| \cdot |\vec{v}_2|} = \frac{x_1 x_2 + y_1 y_2 + z_1 z_2}{\sqrt{x_1^2 + y_1^2 + z_1^2} \cdot \sqrt{x_2^2 + y_2^2 + z_2^2}}$$

Da bei einer Rotation das lokale Koordinatensystem des Globus ebenfalls gedreht wird, muss die Drehachse $\vec{v}_d$ in dieses System überführt werden. Dazu wird aus dem invertierten Rotations-Quaternion der aktuellen Globus-Rotation $-1 \cdot Q_{rot}$ eine Rotationsmatrix M_{rot} abgeleitet.

$$-1 \cdot Q_{rot} = [w, x, y, z]$$

$$M_{rot} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 \end{bmatrix}$$

Durch die Multiplikation dieser Matrix M_{rot} mit dem Vektor der Drehachse $\vec{v}_d$ erhält man den Vektor $\vec{v}_{dl}$ der Drehachse im lokalen Koordinatensystem des Globus .

$$\vec{v}_{dl} = M_{rot} \cdot \vec{v}_d = \cdot \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a_1 x + a_2 y + a_3 z \\ b_1 x + b_2 y + b_3 z \\ c_1 x + c_2 y + c_3 z \end{pmatrix}$$

Aus $\vec{v}_{dl}$ und ω kann jetzt das gesuchte Rotations-Quaternion Q_g errechnet werden.

$$\vec{v}_{dl} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \qquad Q_g = [\cos(\frac{\omega}{2}), \sin(\frac{\omega}{2}) \cdot x, \sin(\frac{\omega}{2}) \cdot y, \sin(\frac{\omega}{2}) \cdot z]$$

Dieses Quaternion wird nun an das RE-Modul übermittelt. Dort wird durch Multiplikation der beiden Quaternionen Q_g und der aktuellen Globus-Rotation die gewünschte Position des Globus erzeugt. Innerhalb der tTHG-Software wird dieser Schritt ebenfalls ausgeführt, da die neue aktuelle Rotation für die Berechnung der Drehachse im nächsten Frame notwendig ist. Die OS-API verfügt zwar über eine Funktion zum Auslesen der aktuellen Rotation, da für die Tests das RE-Modul und die tTHG-Software auf getrennten Rechnern betrieben wurde, erwies sich die Neuberechnung innerhalb der tTHG-Software als performanter.

5.4.2.4.2 Hot-Spot Switch

Die Berechnung der erforderlichen Rotation für die Ausführung der Switch-Geste erfolgt analog zur oben beschriebenen Vorgehensweise. Zusätzlich wird eine Liste aller Kontinente und der dazugehörigen Mittel- bzw. Schwerpunkte benötigt sowie eine Variable, die die Position des aktuell ausgewählten Kontinents in der Liste beinhaltet. Berechnet wird daher eine Rotation bei der der Punkt P die Koordinate des Mittelpunktes des gewünschten Kontinents darstellt und P' für die Position der statischen Berührung steht.

```
L_k = Liste aller Kontinente mit dazugehörigen Mittelpunkten in lat / lon
K_a = Variable aktueller Kontinent

Schleife:

       //Switch-Geste erkannt
       R_a = aktuelle Rotation des Globus
       M_g = Rotationsmatrix des Globus

       //lokale Position des Kontinents K_a + 1
       P = L_k[K_a] lat/lon zu xyz

       //lokale Position der Berührung
       P' = Berührung lat/lon zu xyz
       P'_l = lokale Position = xyz * M_g

       //Quaternion erstellen
       Drehachse und Winkel aus P und P'_l bestimmten
       Quaternion berechnen

       //Rotation an RE
       Sende Text „StoryAddQuaternion w x y z“

       //Kontinent aktualisieren
       K_a = nächster oder erster Eintrag in L_k
```

In der zum Abschluss dieser Arbeit aktuellen Version der tTHG-Software wurde diese Geste allerdings deaktiviert. Grund hierfür ist unter anderem die mangelnde Zuverlässigkeit des cvblob-Algorithmus bei der Erkennung von gleichzeitig stattfindenden Berührungen.

5.5 STATUS DER AKTUELLEN ENTWICKLUNG

In Abbildung 43 bis 46 werden vier Screenshots der tTHG-Software gezeigt. Sie demonstrieren einige der Probleme, die der tTHG aufweist. Abbildung 43 gibt das unbearbeitete Kamerabild wieder. Die fünf, mit gelben Pfeilen markierten, hellen Areale sind das Resultat einer Reflexion von IR-Licht im Bereich der direkten Verlängerung der LEDs. Abbildung 44 zeigt die Maske, die zur Markierung von Bereichen, in denen keine Berührungserkennung möglich ist, eingesetzt wird. Hier sind mit grünen Pfeilen jene Gebiete gekennzeichnet, die durch Reflexionen an den Nahtstellen des Globenkörpers hervorgerufen werden. Alle diese Spiegelungen werden durch die stark reflektierende Innenseite des Globenkörpers begünstigt. Abbildung 45 und 46 beinhalten verschiedene, bearbeitete Versionen der Kamerabilder und jeweils eine Berührung durch eine flach aufgelegte Hand. Die pink gekennzeichnete Reflexion wird durch die Befestigung des Disperser hervorgerufen und markiert dessen Mittelpunkt. Die in Abbildung 43 und 44 blau markierten, verbleibenden Spiegelungen konnten noch nicht zweifelsfrei zugeordnet werden.

Die Detektion von Berührungen gelingt außerhalb dieser angesprochenen Bereiche in der unteren Hemisphäre am ganzen Globus. Die obere Hälfte des Globenkörpers wird allerdings zu schwach ausgeleuchtet, um eine Erkennung zuzulassen. Um die verbleibenden Störungen im Bild zu beseitigen, wird eine Mindestgröße für die Berührungsfläche vorgegeben. Zudem ist die Albedo bzw. der Reflexionsgrad des berührenden Objekts ausschlaggebend für die Detektion. Im Allgemeinen wird eine Berührung mit zwei bis drei Fingern einer Hand erfolgreich erkannt. Bei der Verwendung metallischer Objekte reicht eine Fläche von ca. $2cm^2$ aus.

Außerdem erfolgt eine messbare Reflexion schon bevor ein Objekt die Globusoberfläche berührt. So ist bei der Berührung des Globus mit der ganzen Hand oft ein Teil des Unterarms im Kamerabild sichtbar, obwohl dieser nicht auf der Oberfläche aufliegt.

Daten über die Genauigkeit und Präzision der erkannten Berührungen liefert die Tabelle in Abbildung 41. Für diese Testreihe wurden fünf vorgegebene Punkte mit einem metallischen Zylinder (Berührungsfläche ca. $2cm^2$) berührt und die berechneten Koordinaten für 132 aufeinanderfolgende Frames ausgewertet. Für die Kalibration wurden in diesem Fall vier quadratisch angeordnete Referenzpunkte mit den geographischen Koordinaten R1[0|0], R2[0|-20],R3[20|0] und R4[20|-20] verwendet (siehe Abbildung 42). Die fünf Testpunkte bilden die Seitenmitten und den Mittelpunkt dieses Quadrats (P1[0|-10], P2[10|-20], P3[20|-10], P4[10|0] und P5[10|-10]). Die durchgängig niedrigen Werte der Standardabweichung und Varianz zeigen, dass die Berechnung der Berührungskoordinaten präzise erfolgt. Bei der Betrachtung der Genauigkeit fällt jedoch auf, dass die Bestimmung der Breite für Punkte deren Breite

nicht direkt durch einen Referenzpunkt kalibriert wird (P1, P3 und P5) mit einer nicht unerheblichen Abweichung von ~ 2° erfolgt. Dieser Fehler ist auf die lineare Interpolation bei der Bestimmung der Koordinaten in Zusammenhang mit der konvexen Form des Disperses zurückzuführen.

	Berührung	Berechneter Wert*	Abweichung zur Berührung	Standard-Abweichung	Varianz
P1					
Länge	0	0.003	-0.003	1.00E-03	1.01E-06
Breite	-10	-8.196	-1.804	1.11E-01	1.24E-02
P2					
Länge	10	9.751	0.249	7.27E-02	5.28E-03
Breite	-20	-19.981	-0.019	3.69E-03	1.36E-05
P3					
Länge	20	20.000	0.000	0.00E+00	0.00E+00
Breite	-10	-7.842	-2.158	1.15E-01	1.33E-02
P4					
Länge	10	10.114	-0.114	5.59E-02	3.12E-03
Breite	0	0.000	0.000	0.00E+00	0.00E+00
P5					
Länge	10	10.318	-0.318	8.64E-02	7.47E-03
Breite	-10	-7.926	-2.074	7.82E-02	6.11E-03

* Mittel aus 132 Frames

Abbildung 41: Präzision der Berührungserkennung (Quelle: eigene Darstellung)

Diese ungenaue Bestimmung hat jedoch nur einen geringen Einfluss auf die Bedienung des tTHG. Bei der Durchführung einer einfachen Rotation wird die Drehung des Globus für jeden Frame berechnet und angepasst. Unter Berücksichtigung der in den Tests erreichten Bildwiederholraten im Bereich zwischen 20 und 30 Frames pro Sekunde, ergibt sich daraus eine Aktualisierung der Rotation nach 33 bis 50 Millisekunden. Diese kurze Verzögerung wird kaum bewusst wahrgenommen. Der Nutzer erhält so in Echtzeit ein visuelles Feedback auf seine Eingabe und passt die Bewegung dieser Rückmeldung an.0.00mm

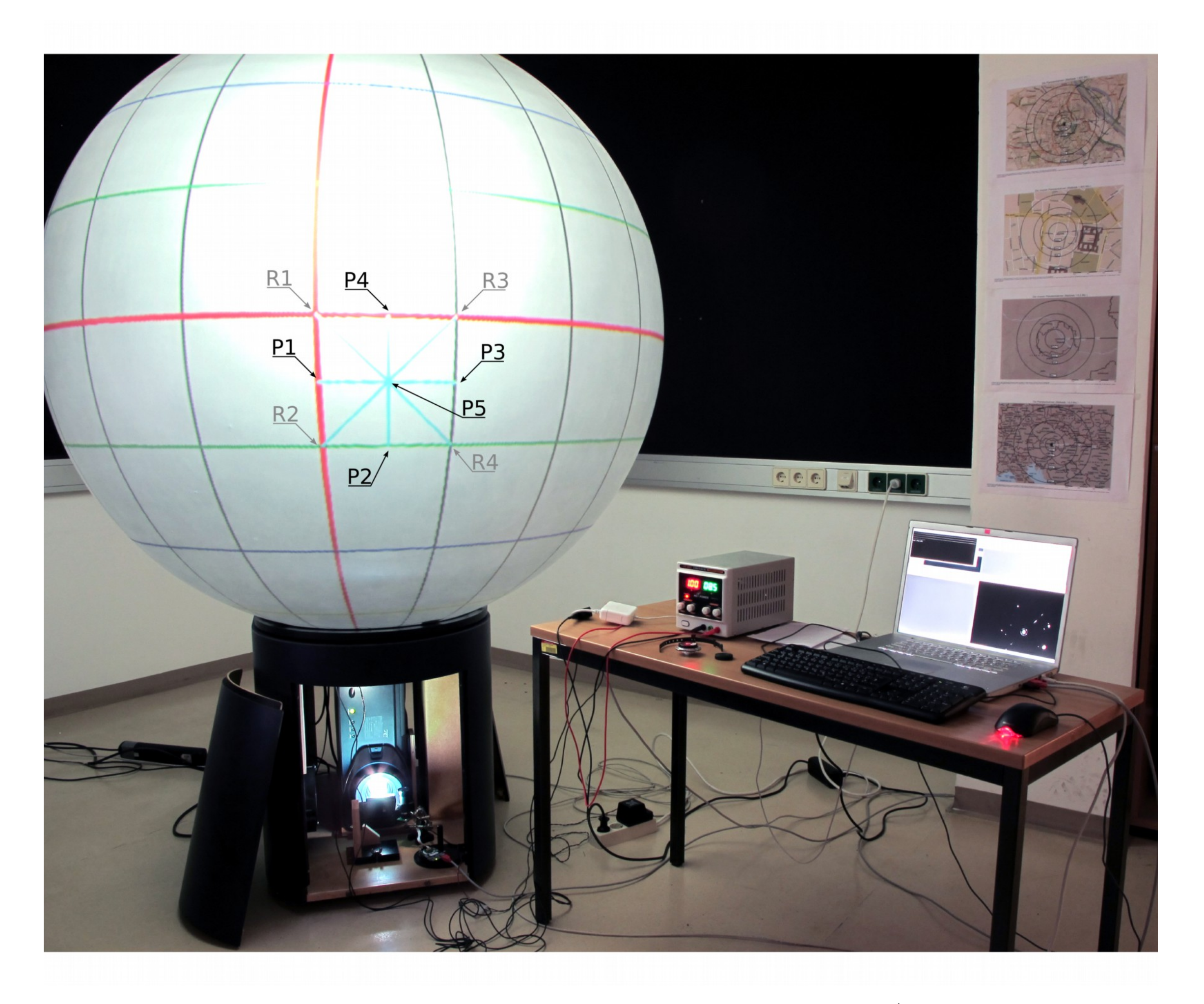
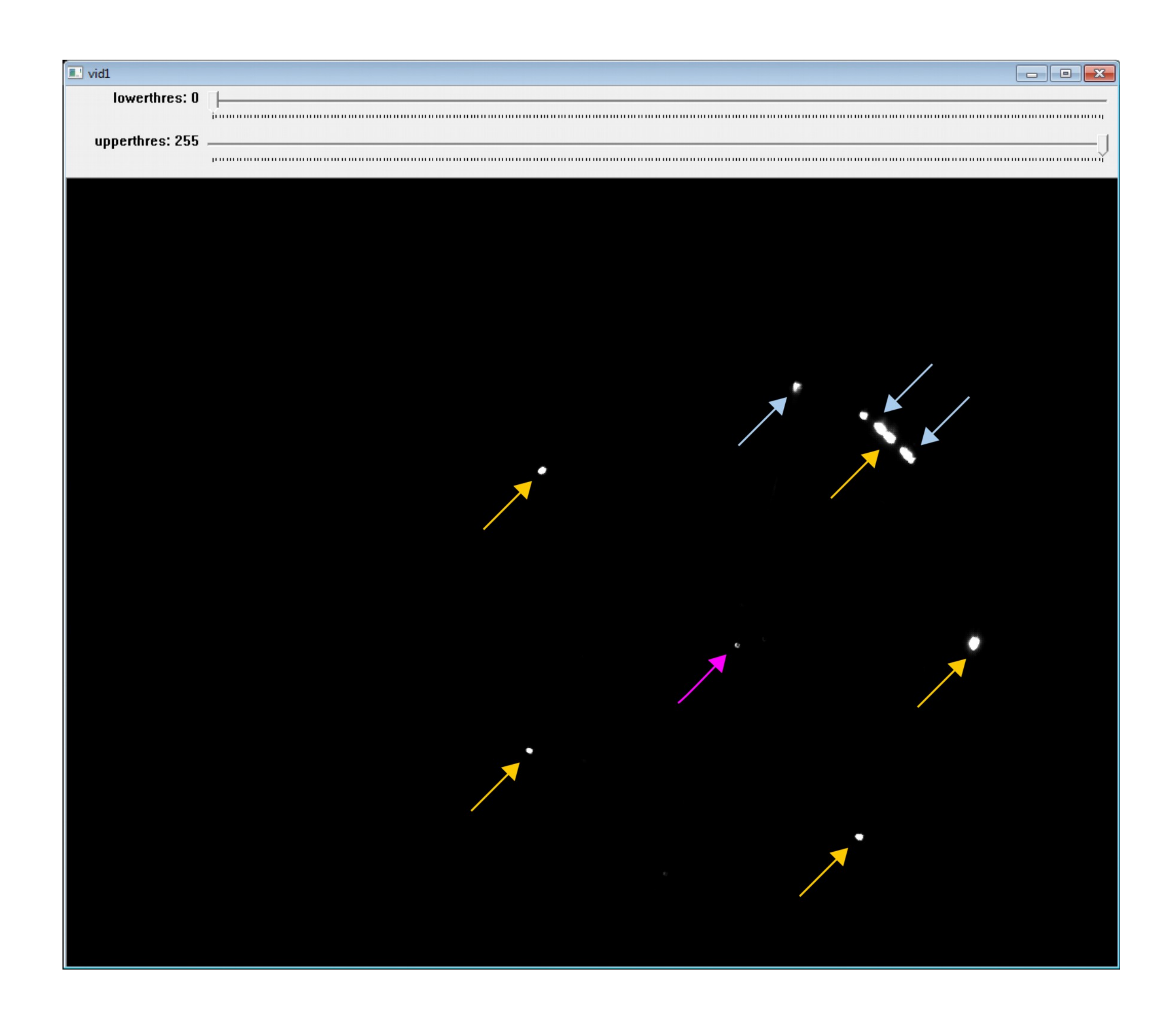


Abbildung 42: tTHG Versuchsaufbau, Referenzpunkte R1 - R4 und Testpunkte P1 - P5 (Quelle: eigene Darstellung)

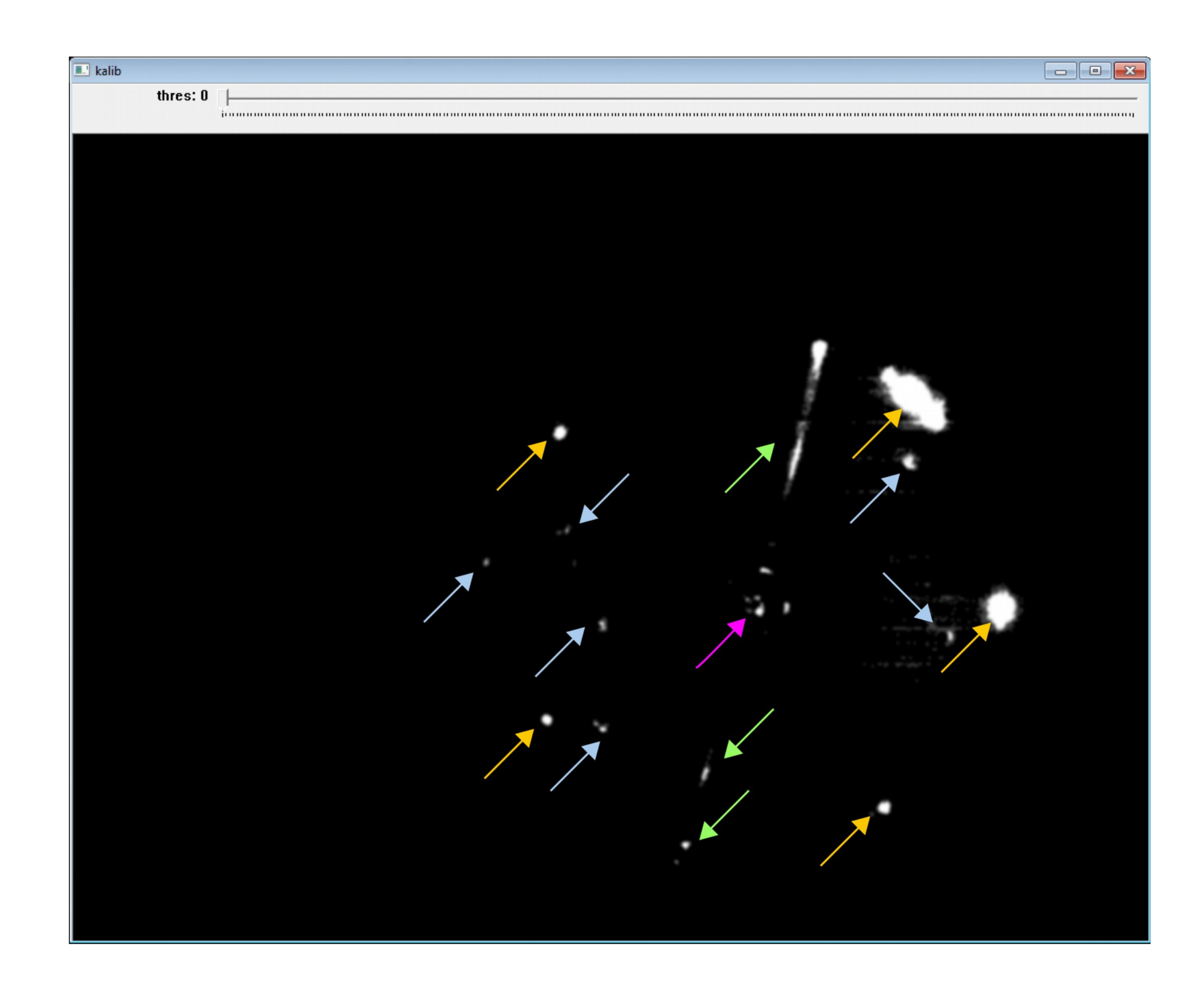


IR-LED, direkte Reflexion

Disperser, Mittelpunkt

nicht identifizierte Reflexion

Abbildung 43: tTHG-Software Screenshot, Rohdaten (Quelle: eigene Darstellung)



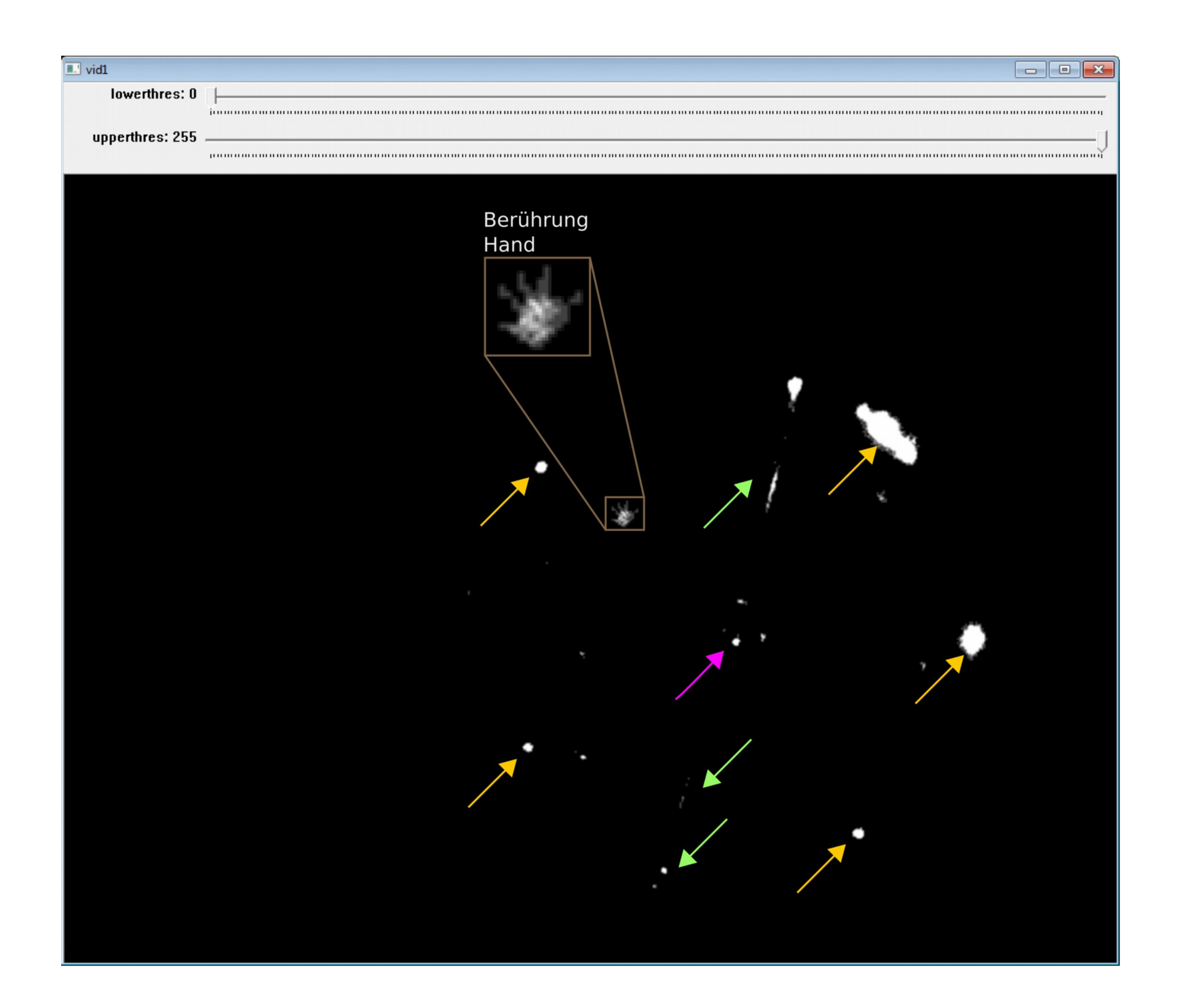
IR-LED, direkte Reflexion

Disperser, Mittelpunkt

nicht identifizierte Reflexion

Globenkörper, Nahtstelle

Abbildung 44: tTHG-Software Screenshot, Maske (Quelle: eigene Darstellung)

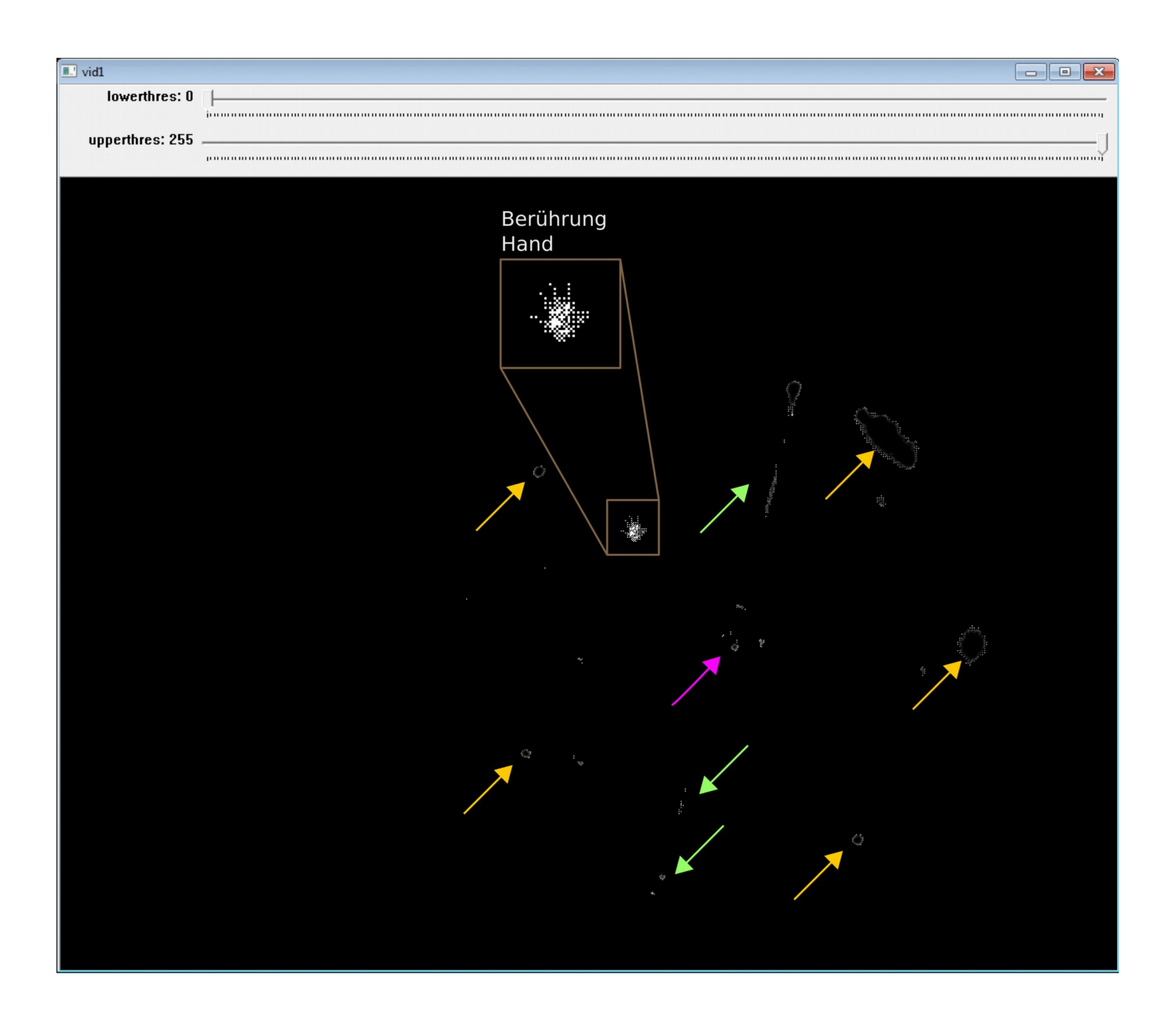


IR-LED, direkte Reflexion

Disperser, Mittelpunkt

Globenkörper, Nahtstelle

Abbildung 45: tTHG-Software Screenshot, Bild nach Anwendung einer Histogrammspreizung und eines Weichzeichnungsfilters (Quelle: eigene Darstellung)



IR-LED, direkte Reflexion

Disperser, Mittelpunkt

Globenkörper, Nahtstelle

Abbildung 46: tTHG-Software Screenshot, Bild nach Anwendung einer Histogrammspreizung und der Kalibrationsmaske (Quelle: eigene Darstellung)

6. AUSBLICK

Bis jetzt konnte gezeigt werden, dass die Erweiterung eines taktilen Hyperglobus zu einem tTHG-Prototypen mit relativ einfachen Maßnahmen durchführbar ist. Jedoch sind die Möglichkeiten, die eine Berührungs- und Gestensteuerung bietet, mit einer einfachen Rotation bei weitem noch nicht ausgeschöpft. Um das volle Potential eines solchen Konzepts nutzen zu können, sind allerdings einige Optimierungen notwendig.

Ein offensichtlicher Schwachpunkt der aktuellen Umsetzung ist die Ausleuchtung des Globenkörpers durch die IR-LEDs. Obwohl bei der Auswahl der LEDs auf eine geeignete Abstrahlcharakteristik Wert gelegt wurde, erweist sich das von ihnen abgegebene Licht als zu fokussiert. Diese Bündelung des Lichts ist vor allem im Zusammenhang mit der reflektierenden Innenseite des Globenkörpers problematisch, da sie großflächige Störungen im Kamerabild verursacht. Im Folgenden werden einige Maßnahmen besprochen die zu einer verbesserten Ausleuchtung führen könnten:

- Anzahl der LEDs:
  Durch den Einsatz von Standard-LEDs, anstatt der in der aktuellen Version verwendeten High-Power-LEDs, würde die abgegebe Strahlung in kleinere Einheiten unterteilt und eine Konzentration des Lichts auf die Bereiche vor den LEDs vermindert werden. Um dieselbe Lichtmenge zu erzeugen, muss jedoch eine sehr große Anzahl verbaut werden. Die momentan eingesetzten LEDs weisen eine Leistungsaufnahme von insgesamt ca. 8 W auf. Um diese Leistung mit Standard-LEDs zu erreichen, die im Bereich von ~50 mW anzusiedeln sind, müssten ca. 160 einzelne LEDs an der Ghost-trap befestigt werden.

- Positionierung:
  Wie in Abbildung 34 zu erkennen ist, wurden alle IR-LEDs flach aufliegend an der Seite der Ghost-trap mit Hilfe von Magneten befestigt, sodass die zentrale Achse der LEDs in einem rechten Winkel zur Seite der Ghost-trap liegt. Für eine gleichmäßige Ausleuchtung könnte sich eine Veränderung des Winkels einiger LEDs als vorteilhaft erweisen. Allerdings muss sichergestellt werden, dass dies zu keiner direkten Beleuchtung des Dispersers führt.

- Diffusion:
  Eine weitere Option ist die Verwendung von Materialien, die passierendes Licht streuen bzw. diffundieren und direkt vor der LED angebracht werden. Diese Materialien können aus speziellen Folien, Kunststoffgehäusen oder Linsen bestehen. Die Verwendung solcher Diffusoren war für die Umsetzung des tTHG vorgesehen, konnte jedoch aufgrund der ein-

geschränkten Möglichkeiten hinsichtlich der Montage solcher Bauteile nicht durchgeführt werden.

Einen weiteren Bereich zur Optimierung des Systems stellt die eingesetzte Kamera dar. Für eine zeitgemäße Multitouch-Steuerung ist die Erfassung von Berührungen, die durch einzelne Finger hervorgerufen werden, Voraussetzung. Um die Mindestgröße der erfassten Objekte zu verringern, ist neben der Vermeidung von Störungen durch die LEDs eine Erhöhung der geometrischen Auflösung und Sensibilität nötig. Die Anzahl der Bildpixel lässt sich unkompliziert durch die Verwendung eines entsprechenden CCD-Chips steigern. Dabei muss beachtet werden, dass sowohl die Vorverarbeitung als auch die Auswertung höher aufgelöster Bilder mehr Rechenleistung bzw. Rechenzeit benötigt. Diesem Umstand kann jedoch durch Anpassungen und Verbesserungen innerhalb der Software zum Teil entgegengewirkt werden. Allerdings reicht eine reine Steigerung der Auflösung nicht aus, um kleinere und schwächere Berührungen zu erkennen. Da die Menge des reflektierten Lichts von der Größe der Berührungsfläche abhängig ist, muss bei kleineren Kontakten sichergestellt werden, dass dieses reflektierte Licht von der Kamera detektiert wird. Eine Möglichkeit dazu ist die Verwendung von lichtstarken Objektiven. Solche weisen einen größeren Eingangsdurchmesser auf und leiten mehr Licht auf den Kamera-Sensor. Auch der Einsatz eine Objektivs mit fester Brennweite könnte dazu beitragen, da solche Modelle in der Regel lichtstärker sind als Zoomobjektive.

Die tTHG-Software ist ebenso wie die tTHG-Hardware als Prototyp zu bezeichnen und weist daher einige Schwachstellen und Verbesserungsmöglichkeiten auf. So könnte die Ausführung beschleunigt werden, indem der Disperser im Bild identifiziert und nur dieser Bereich für die Bildvorverarbeitung und Auswertung berücksichtigt wird. Die OpenCV-Bibliothek unterstützt solche Operationen für Bilder unter dem Schlagwort ROI („Region of interest“). Voraussetzung für die Verwendung dieser Option ist das Vorhandensein einer Maske. Für den Fall des Disperser würde eine exakte Maske eine annähernde Kreisform aufweisen, da eine gewisse Verzerrung aufgrund der Positionierung der Kamera außerhalb der Projektionsachse nicht verhindert werden kann. Eine hinreichend genaue Maske könnte jedoch durch die Berechnung eines Kreises anhand des aus der Kalibration bekannten Mittelpunkts des Dispersers und den Polarkoordinaten desjenigen Referenzpunktes mit der größten nördlichen Breite erstellt werden. Unter der Annahme, dass der Disperser exakt in das Kamerabild eingepasst wird, könnten bei einem Seitenverhältnis des Kamerabilds von 4:3 ca. 40% aller Pixel von der Analyse ausgenommen werden.

In Kapitel 5.5 wurde die Präzision und Genauigkeit der Berührungserkennung besprochen und darauf hingewiesen, dass der Fehler bei der Bestimmung der geographischen Breite für die Ausführung einer einfachen Rotation vernachlässigbar ist. Für weitere Anwendungen ist

dies jedoch nicht der Fall. Wie bereits erwähnt, tritt dieser Fehler durch die lineare Interpolation zwischen den Referenzwerten auf. Für eine korrekte Bestimmung muss jedoch die konvexe Krümmung dieses Spiegels berücksichtigt oder die Dichte der Referenzpunkte erhöht werden. Andererseits sollten im Hinblick auf die Benutzerfreundlichkeit während des Kalibrationsvorgangs so wenig Referenzpunkte wie möglich aufgenommen werden. Da die lineare Interpolation für die Berechnung der Längenwerte ausreichend genaue Werte liefert, sollte das bisher regelmäßige Muster der Referenzpunkte so verändert werden, dass bei gleichbleibender Punktanzahl die Menge der unterschiedlichen Breitenwerte erhöht und die der Längenwerte verringert wird.

Für die Umsetzung einer relativ einfachen Steuerung, die es ermöglicht die Rotation des Globus frei und anhand definierter Punkte (Switch-Geste) zu beeinflussen, reicht die Erkennung von Berührungsflächen aus. Eine komplexere Version, die es erlaubt weitere Funktionen der OS-API und des Controller-Moduls anzusprechen, ließe sich durch eine Erweiterung der Detektion umsetzen, welche nicht nur Flächen sondern auch Formen bzw. Umrisse voneinander unterscheiden kann. Als Beispiel sei hier eine Berührung durch eine ganze Hand mit angelegtem oder abgespreiztem Daumen genannt. Eine solche Erweiterung kann mit Hilfe der in OpenCV bereits implementierten und u.a. aus der Fernerkundung und Photogrammetrie bekannten SIFT („scale invariant feature transform“) und SURF („speeded up robust features“) Algorithmen erfolgen. Mit diesem erweiterten Set von Eingabeaktionen kann eine größere Anzahl von Befehlen aufgerufen werden, ohne auf Kombinationen von nacheinander erfolgenden Berührungen zurückgreifen zu müssen.

Abbildung 47: mögliches On Screen Menu (Quelle: eigene Darstellung)

Eine weitere Möglichkeit für die Auswahl verschiedener Befehle stellt ein „On Screen Menu" (OSM) dar. Abbildung 47 zeigt ein mögliches Beispiel eines solchen Menüs. Dieses Menü wird durch Auflegen einer ganzen Hand mit gespreizten Fingern aktiviert. Daraufhin werden am Globus um diese Berührung Symbole eingeblendet, die sich wie Schaltflächen verhalten und durch einen weiteren Kontakt eine Funktion bzw. ein Werkzeug aktivieren. Allerdings müssen, um solch ein OSM zu ermöglichen, umfangreiche Änderungen bzw. Erweiterungen am RE-Modul und dem Story-Prinzip von OS vorgenommen werden. Voraussetzung für den Einsatz eines OSM sind Funktionen die es ermöglichen nicht nur die Rotation, sondern auch das am Globus dargestellte Bild in Echtzeit zu manipulieren. Dazu könnten die zum Aufbau des OSM erforderlichen Graphiken innerhalb der RE mit der durch die Story vorgegebenen Textur vor dem Rendern überlagert werden.

Eine noch verbleibende Hürde ist dabei die notwendige Verzerrung zu bestimmen, um eine korrekte Darstellung zu gewährleisten, da eine in die Textur (Plattkarte) eingefügte Graphik durch die Umprojektion am Globus verzerrt wiedergegeben wird. LIEM beschreibt dieses Problem und verschiedene Lösungsansätze in [LIE-12]. Für die Selektion und Abfrage von Informationen geographischer Objekte wie Städte, Länder etc. wird eine Verknüpfung zwischen den Textur-Koordinaten und diesen Objekten benötigt. Diese Verknüpfungen könnten in einer Struktur, ähnlich der einer Datenbank, innerhalb des Story- bzw. Materialskripts abgespeichert werden.

Mit der Möglichkeit das Globenbild in Echtzeit zu verändern und beliebige Graphiken und Texte darstellen zu können, wird die Entwicklung von Lernspielen denkbar. Solche Anwendungen entsprechen dem Ziel der HRG, taktile Hypergloben als Instrumente zur Wissensvermittlung zu verwenden. Zum jetzigen Zeitpunkt werden digitale Globen mehrheitlich als Exponate in Museen und Science-Centren eingesetzt. Dabei erfolgt die Vermittlung von Wissen überwiegend auf einem passiven Weg durch geführte Präsentationen von verschiedenen Stories.

Vor allem, weil Kinder und Jugendliche die Hauptzielgruppe solcher Einrichtung darstellen, erscheint der Schritt hin zu einer Form bei der das Publikum bzw. die Nutzer mit eingebunden werden und aktiv teilhaben können sinnvoll. Dies kann durch einfache Aufgaben wie z.B. das Einzeichnen von Grenzverläufen in einer physischen Erddarstellung und einem anschließenden Vergleich mit der realen Situation erfolgen. Eine technisch anspruchsvollere Applikation wäre etwa ein Puzzlespiel, bei dem der Nutzer die zufällig verteilten Länder an ihre richtige Position verschieben soll. Unabhängig von der Komplexität, stellen solche Spiele jedoch eine, in den Augen des Autors, sehr gute Möglichkeit dar globale Sachverhalte und Zusammenhänge spielerisch zu vermitteln.

7. ZUSAMMENFASSUNG

Das Ziel dieser Arbeit besteht darin, die noch junge kartographische Darstellungsform der taktilen Hypergloben weiter zu entwickeln und deren Funktionsumfang durch die Integration einer auf Berührungen und Gesten basierenden Steuerung zu erweitern. Diese Integration erfolgte durch die Entwicklung eines Prototyps (bestehend aus Hard- und Software) , an Hand dessen die prinzipielle Machbarkeit demonstriert und die Grundlagen für weitere Entwicklungen geschaffen wurden.

Um dieses Ergebnis zu erreichen, wurde im ersten Teil der Arbeit ein Überblick über die verschiedenen Arten der digitalen Globen gegeben sowie die technischen Prinzipien und Funktionsweisen von berührungsempfindlichen Eingabegeräten vorgestellt und deren prinzipielle Eignung für den Einsatz an einem THG bewertet. Zudem wurde untersucht auf welche Weise ein solches Steuerungskonzept bei vergleichbaren sphärischen Displays bereits umgesetzt wurde. Auch wenn einige dieser Displays keinen oder nur sehr wenig Bezug zur Darstellung kartographischer Inhalte aufweisen, sind viele der angewandten Prinzipien, wie z.B. die Regeln für die Gestaltung eines Interface, auf Grund der nahezu identischen technischen Grundlagen auf einen digitalen Globus übertragbar.

Die HRG verfügt über einen THG und entwickelt seit einigen Jahren ein sehr umfangreiches Softwarepaket zur Erstellung und Darstellung von Inhalten auf digitalen Globen. Daher war die Verwendung dieses Globus und die Eingliederung der zu erstellenden Anwendung in die bestehende Softwareinfrastruktur naheliegend. An Hand der im zweiten Kapitel gewonnenen Erkenntnisse und der Berücksichtigung der Ausgangssituation wurde ein Konzept zur prototypischen Umsetzung einer solchen Steuerung am THG des IFGR erstellt. Dabei mussten einige besondere Anforderungen berücksichtigt werden, wie etwa die Vorgabe, dass alle durchgeführten Maßnahmen ohne permanente Veränderung des THG erfolgen müssen. Auf der Basis dieses Konzepts wurden die notwendigen Modifikationen an der Hardware durchgeführt. Diese umfassten die Installation einer Infrarotkamera und einiger IR-Leuchtdioden mit deren Hilfe Berührungen des Globus sichtbar gemacht werden. Ergänzend dazu wurde eine Anwendung programmiert mit deren Hilfe die Kamerabilder in Echtzeit ausgewertet und in entsprechende Steuerbefehle übersetzt werden. Während die Auswahl der für die Hardwaremodifikation erforderlichen Komponenten und deren Montage vergleichsweise unkompliziert und schnell erfolgte, stellte die Einarbeitung in die verwendeten externen Bibliotheken und die Programmierung der Software den größten und zeitaufwendigsten Teil der Arbeit dar.

In der zum Abschluss dieser Arbeit aktuellen Version können nach der Durchführung einer Kalibration Berührungen auf ca. 50% der Globenoberfläche erkannt werden. Durch das Zu-

sammenspiel der neu entwickelten Software und dem bereits existierenden OmniSuite Modul „RenderEngine“ ist es möglich, die Rotation des Globus durch Berührungen beliebig zu verändern. Wie bei der Erstellung eines Prototyps zu erwarten, weist die Umsetzung noch einigen Spielraum für Verbesserungen und Optimierungsbedarf auf. Trotzdem konnte gezeigt werden, dass die Konstruktion von berührungsempfindlichen taktilen Hypergloben auch mit geringen Mitteln möglich ist. Darüber hinaus, müssen bei der Bewertung dieser Resultate die strikten Vorgaben sowie der begrenzte technische und finanzielle Rahmen des Projekts mit in Betracht gezogen werden.

Im letzten Teil der Arbeit wurden unter anderem verschiedene Ansätze und Strategien zur Optimierung des Systems besprochen. Dies betrifft insbesondere eine Vergrößerung des berührungsempfindlichen Bereichs am Globus durch Veränderung im Aufbau der Hardwarekomponenten. Ebenso wurde diskutiert mit welchen Mitteln eine komplexere und effizientere Gestaltung der Berechnungen innerhalb der Software erfolgen kann und welchen Beitrag die durch den Prototypen gewonnenen Erkenntnisse zur Entwicklung weiterer Funktionen und Anwendungen leisten können. Vor allem in den angesprochenen Lernspielen vermutet der Autor großes Potenzial, da sie den THG in ein höchst interaktives Instrument zur Wissensvermittlung verwandeln könnten. Auch wenn auf dem Weg dahin noch einige Probleme überwunden werden müssen, kommt der Autor zu dem Schluss, dass eine Berührungssteuerung eine positive und sinnvolle Erweiterung des THG darstellt und in Zukunft wohl zur Grundausstattung taktiler Hypergloben gehören wird.

QUELLEN

[BEN-08]

BENKO Hrvoje, WILSON Andrew D. , BALAKRISHNAN Ravin (2008): Sphere: Multi-Touch Interactions on a Spherical Display; http://research.microsoft.com/en-us/um/people/benko/publications/2008/Benko_Sphere_UIST2008.pdf (zuletzt abgerufen am 5.8.2012)

[BOL-11]

BOLTON John, KIM Kibun, VERTEGAAL Roel (2011): SnowGlobe: A Spherical Fish-Tank VR Display; http://www.hml.queensu.ca/files/sphere.pdf (zuletzt abgerufen am 5.8.2012)

[BRA-08]

BRADSKI Gary, KAEHLER Adrian (2008): Learning OpenCV – Computer Vision with the OpenCV Library; O'Reilly Media; Köln; 580 S.

[BRO-08]

BRONSTEIN Ilija N. et al. (2008): Taschenbuch der Mathematik; 7. Auflage; Harri Deutsch; Frankfurt a.M.; 1216 S.

[CVB-12]

cvblob – Blob library for OpenCV (2012); http://code.google.com/p/cvblob/(zuletzt abgerufen am 01.10.2012)

[ERT-06]

ERTUGRUL, Murat (2006): Erläuterung des API-Ansatzes; Studienarbeit; GRIN Verlag; München; 15 S.

[HAK-94]

HAKE Günter, GRÜNREICH Dietmar (1994): Kartographie; 7. Auflage; Berlin; de Gruyter; 599 S.

[HAL-10]

HALBRITTER, Hubert(2010): Emiters and Detectors for Infrared (IR) Touchscreens – Application note; Osram Opto Semiconductors GmbH; Regensburg; http://catalog.osram-os.com/catalogue/catalogue.do;jsessionid=5E2409210BA74156917717AC676D7667?act=downloadFile&favOid=0200000400012396000200b6 (zuletzt abgerufen am 29.09.2012)

[HML-11]

HUMAN MEDIA LABS (2011): DIY Spherical Display Plans; http://www.humanmedialab.org/node/302 (zuletzt abgerufen am 14.08.2012)

[HRU-09]

HRUBY Florian (2009): Der digitale Globus – Begriff und Bedeutung für die Geographie; in: KRIZ K, KAINZ W., RIEDL A. (Hrsg.): Geokommunikation im Umfeld der Geograohie; Wiener Schriften zur Geographie und Kartographie; Band 19; Wien; Universität Wien – Institut für Geographie und Regionalforschung

[IFGR-12]

Institut für Geographie und Regionalforschung, Universität Wien (2012); Arbeitsgruppe Kartographie und Geoinformation; http://carto.univie.ac.at/forschungsprojekte (zuletzt abgerufen am 17.08.2012)

[KRE-86]

KRETSCHMER Ingrid, DÖRFLINGER Johannes, WAWRIK Johannes Franz (1986): Lexikon zur Geschichte der Kartographie, in: Die Kartographie und ihre Randgebiete, Bd. C/1 und C/2; 2 Teilbände; Deuticke; Wien

[KRI-12]

KRISTEN Jürgen (2012): 3D-Grafikprogrammierung interaktiver kartographischer Echtzeit-Anwendungen — am Beispiel eines taktilen Hyperglobus; Diplomarbeit; Universität Wien; Wien

[LEN-11]

LENK Ron, LENK Carol (2011): Pracitcal Lighting Design with LEDs; John Wiley and Sons Ltd.; Hoboken (New Jersey, USA); 272 S.

[LIE-12]

LIEM Johannes (2012): Problemfelder und Lösungsansätze bei der Wiedergabe von geographischen Vektordatensätzen am sphärischen Display; Diplomarbeit; Universität Wien; Wien

[MÖS-11]

MÖSTEL Stefan (2001): Multimodale Systeme im mobilen und stationären Einsatz : Am Beispiel der kollaborativen Geschäftsprozessmodellierung; Bachelorarbeit; GRIN Verlag; München; 148 S.

[ODE-10]

ODENDAHL Manuel, FINN Julian, WENGER Alex (2010): Arduino - Physical Computing für Bastler, Designer und Geeks.; O'Reilly Media; Köln; 400 S.

[OPC-12]

OpenCV - Open Source Computer Vision (2012); http://opencv.willowgarage.com/wiki/ (zuletzt abgerufen am 2.10.2012)

[OPC-12a]

OpenCV – OpenSource Computer Vision (2012): Documentation; http://opencv.willowgarage.com/documentation/cpp/ (zuletzt abgerufen am 2.10.2012)

[OSR-12]

OSRAM Opto Semiconductors GmbH (2012): SFH4235 Datasheet; Osram Opto Semiconductors GmbH; Regensburg;http://catalog.osram-os.com/catalogue/catalogue.do;jsessionid= 7690E4E80173310AF60013F7D7A11C6F?act=downloadFile&favOid=020000000000179370003 00b6 (zuletzt abgerufen am 29.09.2012)

[PUF-12]

PUFFERFISCH (2012): Expert – spherical projection; http://www.pufferfishdisplays.co.uk (zuletzt abgerufen am 17.08.2012)

[RIE-00]

RIEDL Andreas (2000): Virtuelle Globen in der Geovisualisierung, Untersuchungen zum Einsatz von Multimediatechniken in der Geopräsentation ; Dissertation ; Universität Wien; http://homepage.univie.ac.at/andreas.riedl/pub/2000_digital_globes_hyberglobes.pdf; (zuletzt abgerufen am 28.06.2012)

[RIE-08]

RIEDL Andreas (2008): Entwicklung und Perspektiven von Taktilen Hypergloben. (Preprint); in: Mitteilungen der Österreichischen Geographischen Gesellschaft; Band 150; Wien

[RIE-10]

RIEDL, Andreas (2010): Entwicklungsgeschichte digitaler Globen; http://carto.univie.ac.at/uploads/media/6_2010_GF_digitaleGloben_Riedl.pdf (download am 17.08.2012)

[RIE-11]

RIEDL Andreas (2011): Der Globus ist tot, es lebe der Globus!; in: KRIZ K., KAINZ W., RIEDL A. (Hrsg.): 50 Jahre Österreichische Kartographische Kommission. Wien, Institut für Geographie der Universität Wien, 2011 (=Wiener Schriften zur Geographie und Kartographie, Band 20) S. 79 - 87.

[SAF-08]

SAFFER Dan (2008): Designing Gestural Interfaces, Touchscreens and Interactive Devices; O'Reilly Media; Köln: 272 S.

[SNE-06]

SNEHI Jyoti (2006): Computer Peripherals and Interfacing; Laxmi Publications; New Delhi; 136 S.

[SON-12]

SONY Corporation (2012): ICX204AL Datasheet; http://www.sony.net/Products/SC-HP/datasheet/90203/data/a6811241.pdf (zuletzt abgerufen am 29.09.2012)

[UNI-12]

UNIBRAIN S.A (2012): Unibrain Firewire (1394a) IIDC 1.31 OEM XGA board camera; http://www.unibrain.com/products/visionimg/boardxga.htm (zuletzt abgerufen am 29.10.2012)

[UNI-12a]

UNIBRAIN S.A (2012): Fire-i API Specification; http://www.unibrain.com/download/pdfs/SDK/FireiAPISpec.chm (zueltz abgerufen am 02.12.2012)

[WEB-12]

WEBB Jarrett, ASHLEY James (2012): Beginning Kinect Programming with the Microsoft Kinect SDK; Berlin; Springer; 324 S.

[WIT-79]

WITT Werner (1979): Lexikon der Kartographie, in: Die Kartographie und ihre Randgebiete,Bd. B; Wien; Deuticke; 707 S.

Anhang

Quelltext: beta.cpp

```
// beta.cpp : Defines the entry point for the console application.
//


#define _WINSOCKAPI_
#include <winsock2.h>

#include "my_ubcore.h"



#include <stdio.h>
#include <tchar.h>

#include <iomanip>



#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>

#include <opencv2\opencv.hpp>
#include <opencv\highgui.h>
#include <opencv\cxcore.h>
#include <opencv\cv.h>

#include <cvblob.h>


#include <windows.h>
#include <FIREI.H>


#include "kalib_functions_and_types.h"
#include "kreis_berechnungen.h"
#include "Referenzgitter.h"
#include "Referenzpunkt.h"
#include "blob_functions.h"


using namespace std;
using namespace cvb;


//Prototypen
//int startWinsock(void);
int startWinsock()

{
  WSADATA wsa;
  return WSAStartup(MAKEWORD(2,0),&wsa);
 }


//int _tmain(int argc, _TCHAR* argv[])
int main(int argc, _TCHAR* argv[])
{

        ////////////////////////////////////////////////////////////////////////////////
        //some windows magic to move the console window
        char title[500];  // to hold title
        // get title of console window
        SetConsoleTitleA("console");
        GetConsoleTitleA( title, 500 );
```

```
    // get HWND of console, based on its title
    HWND hwndConsole = FindWindowA( NULL, title );

    // get HINSTANCE of console, based on HWND
    HINSTANCE hInstance = (HINSTANCE)GetWindowLong(hwndConsole, GWL_HINSTANCE);

    SetWindowPos(hwndConsole,HWND_TOP,0,0,500,200,SWP_NOSIZE);
    //////////////////////////////////////////////////////////////////////////////////

    //#####################################################################################
    //===============================
    // -> START THE CAMERA

    FIREi_CAMERA_HANDLE  g_hCamera ;
    FIREi_ISOCH_ENGINE_HANDLE   g_hIsochEngine;


    FIREi_CAMERA_STARTUP_INFO   g_StartupInfo;

    FIREi_STATUS FiStatus;

    FIREi_CAMERA_GUID   CameraGuid;
    ULONG               uCamerasFound;




    FIREi_CAMERA_FRAME  CameraFrame;




    //initialize
FiStatus = FiInitialize();

if ( FiStatus != FIREi_STATUS_SUCCESS )
{
        printf("FiInitialize failed with status code %u\n", FiStatus);
        return -1;
}


    uCamerasFound = 1;

    //search for cameras
FiStatus = FiLocateCameras(&CameraGuid,FIREi_LOCATE_NEW_CAMERAS,&uCamerasFound );
if(FIREi_STATUS_SUCCESS != FiStatus)
{
    printf("FiLocateCameras failed with status code %u\n", FiStatus);
        return -2;
}


    //get camera handle
    FiStatus = FiOpenCameraHandle( &g_hCamera, &CameraGuid);
    if ( FiStatus != FIREi_STATUS_SUCCESS )
{
       printf("FiOpenCameraHandle failed with status code %u\n", FiStatus);
       return -3;
}


    // 800x600@30FPS = Mode_2 Format_1 S400
    // 640x480@30FPS = Mode_5 Format_0 S200
    // 1024x768@30FPS = Mode_5 Format_1

g_StartupInfo.Tag = FIREi_CAMERA_STARTUP_INFO_TAG;
g_StartupInfo.FrameRate     = fps_30;
g_StartupInfo.VideoMode     = Mode_5;
g_StartupInfo.VideoFormat   = Format_1;
g_StartupInfo.TransmitSpeed = S400;
g_StartupInfo.IsochSyCode   = 1;
g_StartupInfo.ChannelNumber = (UCHAR)9;
```

```
        FiStatus = FiStopCamera(g_hCamera);
        if ( FiStatus != FIREi_STATUS_SUCCESS )
    {
            printf(" FiStopCamera failed with status code %u\n", FiStatus);
                //return -4;
    }


        //stop camera
        FiStatus = FiStartCamera( g_hCamera, &g_StartupInfo);
        if ( FiStatus != FIREi_STATUS_SUCCESS )
    {
            printf(" FiStartCamera failed with status code %u\n", FiStatus);
            return -4;
    }


        //create IsoRecieveEngine
        FiStatus = FiCreateIsochReceiveEngine(&g_hIsochEngine);
        if ( FiStatus != FIREi_STATUS_SUCCESS )
    {
            printf(" FiCreateIsochReceiveEngine failed with status code %u\n", FiStatus);
            return -5;
    }


        //start isoRecieveEngine
        FiStatus = FiStartIsochReceiveEngine(g_hIsochEngine, &g_StartupInfo, 1);
        if ( FiStatus != FIREi_STATUS_SUCCESS )
    {
            printf(" FiStartIsochReceiveEngine failed with status code %u\n", FiStatus);
            return -6;
    }

        // END START THE CAMERA <-
        //==============================










        //#####################################################################################
        //==============================
        // -> define variables

                //OpenCV Stuff
                CvSize size = cvSize(1024,768);
                int depth = IPL_DEPTH_8U;
                int channels = 1;



                IplImage *frame_bw1 = cvCreateImage(size,IPL_DEPTH_8U,channels); // main image
                IplImage *frame_kalib = cvCreateImage(size,IPL_DEPTH_8U,channels);        // image
for masking
                IplImage *mp_overlay = cvCreateImage(size,IPL_DEPTH_8U,channels); // multi purpose
overlay
                IplImage *canny = cvCreateImage(size,IPL_DEPTH_8U,channels);      // multi purpose
overlay



                CvFont font;
                cvInitFont(&font, CV_FONT_HERSHEY_SIMPLEX, 1.0, 1.0, 0, 1, CV_AA);

                //BLOB STUFF
                CvTracks tracks;
```

```
//
int kalib_thres = 255; // values above this int will be set to 255 in kalib_image
int gaus = 3; // smooth gaus value
int minblobsize = 8;
int maxblobsize = 150;
int vthres = 0;
int uvthres = 255;

double old_lat = 0;
double old_long = 0;

// Quaternion das die Roatation des letzten Frames beinhaltet
quaternion old_rot;
old_rot.w = 1;
old_rot.x = 0;
old_rot.y = 0;
old_rot.z = 0;


double track_static_threshold = 2.5; // threshold between static and dynamix tracks



//Image processing loop control variables
bool quit = false;
bool mask = false;
bool use_kalib = true;
bool pause = false;
bool use_smooth = false;
bool use_blob = false;
bool update_vid = true;
bool use_equal = false;
bool quit_kalib = false;
bool use_canny = false;
bool use_vthres = false;
bool use_uvthres = false;
bool use_lines = false;
bool draw_kreis = false;
bool calc_globe_coord = false;
bool use_rotation = false;

//counters
int missed_frames = 0; // frames that couldn't be transfered within time limit
int first_frames = 0;  // counter used for mask creation
int total_frames = 0;  // total # of frames

//ROI
double xmin = 999999;
double ymin = 999999;
double xmax = 0;
double ymax = 0;

bool roi_change = false;

//draw lines
int input = 0;
line l1;
line l2;
line lines[2] = {l1,l2};
char str[2];

//draw circle
struct mPunkt p1;
p1.x = 0;
p1.y = 0;
struct mPunkt p2;
p2.x = 0;
p2.y = 0;
struct mPunkt p3;
p3.x = 0;
p3.y = 0;
struct mPunkt m;
m.x = 0;
m.y = 0;
```

```
		int p_counter = 0;
		struct mKreis kreis;
		kreis.mittelpunkt = m;
		kreis.radius = 0;

		zentrum centre;

		CvBlobs blobs;

	// end variables definition <-
	//===============================




	//##########################################################################################
	//===============================
	// -> tcp/ip network socket startup

	// initialize
	WSADATA wsa;
	long    rc;
	SOCKET  s;
	SOCKADDR_IN Player_addr;
	memset(&Player_addr,0,sizeof(SOCKADDR_IN));
	Player_addr.sin_family=AF_INET;
	Player_addr.sin_port=htons(67890);
	Player_addr.sin_addr.s_addr=inet_addr("131.130.52.211");
	WSAStartup(MAKEWORD(2,0),&wsa);
	rc= startWinsock();
	if(rc!=0)
	{
		cout << "error in socket init" << endl;
		return -11;
	}
	s=socket(AF_INET,SOCK_DGRAM,0);
	if(s==INVALID_SOCKET)
	{
		cout << "error in socket init2" << endl;
		return -12;
	}


	// End Socket startup <-
	//===============================


	// DEBUG -> SELECT STORY
	std::ostringstream strs1;
	strs1 << "StoryOpen touchtest1";
	std::string message1 = strs1.str();
	rc = sendto(s, message1.c_str(), strlen(message1.c_str()), 0, (SOCKADDR*)&Player_addr,
sizeof(SOCKADDR_IN));




	//##########################################################################################
	//===============================
	// -> OpenCV gui stuff


		//create a window
		cvNamedWindow( "vid1", CV_WINDOW_AUTOSIZE );
		cvMoveWindow("vid1",0,350);

		cvNamedWindow("blob", CV_WINDOW_AUTOSIZE);
		cvMoveWindow("blob",810,0);
		cvCreateTrackbar( "blob_min", "blob", &minblobsize, 100, 0 );
		cvCreateTrackbar( "blob_max", "blob", &maxblobsize, 5000, 0 );
		cvCreateTrackbar( "lowerthres","vid1", &vthres, 255, 0 );
		cvCreateTrackbar( "upperthres","vid1", &uvthres, 255, 0 );

	// End OpenCV Gui Stuff <-
```

```
//================================






//#####################################################################################
//================================
// -> get first frames for adjustment images


blind_spot_kalib (&CameraFrame, &g_hIsochEngine, 30, frame_kalib);

cvDestroyWindow("kalib");

// End get first frames for adjustment images <-
//================================




//#####################################################################################
//================================
// -> Setting up refrence grid

Referenzgitter refgrid;
vector<Referenzpunkt> ref_list;
sorted_ll_list sort;
CvTracks ref_tracks;



// End setting up refrence grid <-
//================================


//#####################################################################################
//================================
// -> Image Processing Loop

//cvDestroyWindow("vid1");


while (!quit)
{

        //get frame
        if (!pause)
        {
                if (get_frame(&CameraFrame, &g_hIsochEngine, 30 , frame_bw1) == 1)
                {
                        missed_frames += 1;
                        printf ("frame missed\n");
                        continue;
                }
        }

        //toggle useage of kalib image
        if (use_equal) cvEqualizeHist(frame_bw1,frame_bw1);

        if (use_smooth) cvSmooth(frame_bw1, frame_bw1, CV_GAUSSIAN, gaus);

        if (use_canny)
        {
                cvCanny(frame_bw1,canny,10,200);
                cvCopy(canny,frame_bw1);
        }
        if (use_vthres)
        {
                cvThreshold(frame_bw1,frame_bw1,vthres,255,CV_THRESH_TOZERO);
                cvThreshold(frame_bw1,frame_bw1,uvthres,255,CV_THRESH_TOZERO_INV);
        }
```

```
		if (use_kalib) cvSub(frame_bw1,frame_kalib,frame_bw1);


		// simple blob detection
		if (use_blob)
		{
			// local variables
			IplImage *labelImg=cvCreateImage(size, IPL_DEPTH_LABEL, 1);
			IplImage *frame_col=cvCreateImage(size, IPL_DEPTH_8U, 3);

			unsigned int result;

			if (roi_change)
			{
				//cvSetImageROI(frame_bw1,cvRect(xmin,ymin,(xmax-xmin),(ymax-
ymin)));
			}


			// analyze image and get blobs
			result=cvLabel(frame_bw1, labelImg, blobs);

			cvResetImageROI(frame_bw1);

			// Filter Blobs
			//cvFilterByArea(blobs, minblobsize, 10 * maxblobsize);
			if (minblobsize == 0) minblobsize = 1;
			cvFilterByArea(blobs, minblobsize, 10 * maxblobsize);

			//Tracks
			cvUpdateTracks(blobs, tracks, 10., 2);

				if (use_rotation)
				{
				// Tracks bearbeiten
				for (CvTracks::iterator it = tracks.begin(); it != tracks.end(); it
++)
				{
					// Zentrum im ersten frame setzen
					if ( it->second->lifetime <= 1)
					{
						it->second->init_x = it->second->centroid.x;
						it->second->init_y = it->second->centroid.y;
						it->second->max_dist = 0;
						double m10,m01,m00 = 0;
						//Blob finden
						for (CvBlobs::iterator bt = blobs.begin(); bt !=
blobs.end(); bt ++)
						{
							if (it->second->label == bt->second-
>label)
							{
								m10 = bt->second->m10;
								m01 = bt->second->m01;
								m00 = bt->second->m00;
							}
						}
						it->second->init_schwerpunkt.x = m10 / m00;
						it->second->init_schwerpunkt.y = m01 / m00;
					}

					//Schwerpunkt bestimmen
					double m10,m01,m00 = 0;

						//Blob finden
						for (CvBlobs::iterator bt = blobs.begin(); bt !=
blobs.end(); bt ++)
						{
							if (it->second->label == bt->second-
>label)
							{
								m10 = bt->second->m10;
								m01 = bt->second->m01;
								m00 = bt->second->m00;
							}
```

```
                                        }



                                // lat/lon of init-centroid
                                vector<double> gk_init;
                                passpunkt p_init;
                                p_init.pic_x = it->second->init_x;
                                p_init.pic_y = it->second->init_y;
                                gk_init = refgrid.calc_lat_long(p_init);


                                // lat/lon of current centroid
                                vector<double> gk_act;
                                passpunkt p_act;
                                p_act.pic_x = it->second->centroid.x;
                                p_act.pic_y = it->second->centroid.y;
                                gk_act = refgrid.calc_lat_long(p_act);

                                // calculate distance to init centroid
                                // update if needed
                                double b_a = rad(gk_init[0]); //breite punkt a in radiant
                                double l_a = rad(gk_init[1]); //laenge punkt a in radiant
                                double b_b = rad(gk_act[0]);
                                double l_b = rad(gk_act[1]);
                                double dist = deg(acos( sin(b_a)*sin(b_b) +
cos(b_a)*cos(b_b)*cos(l_b-l_a))); //calculate orthodrome

                                //DEBUG
                                //cout << "id: " << it->second->id << " dist: " << dist <<
" max_dist: " << it->second->max_dist << endl;

                                if (it->second->max_dist < dist) it->second->max_dist =
dist;

                                // distinguish static/dynamic track
                                it->second->statisch = ( it->second->max_dist >
track_static_threshold) ? false : true;


                                // draw static/dynamic tracks
                                //if (it->second->statisch) cvRectangle(frame_col,
cvPoint(it->second->minx, it->second->miny), cvPoint(it->second->maxx-1, it->second->maxy-1),
CV_RGB(255., 0., 0.),4);
                                //if (! it->second->statisch) cvRectangle(frame_col,
cvPoint(it->second->minx, it->second->miny), cvPoint(it->second->maxx-1, it->second->maxy-1),
CV_RGB(0., 0., 255.),4);
                        }
                        }


                cvRenderTracks(tracks, frame_col, frame_col, CV_TRACK_RENDER_ID|
CV_TRACK_RENDER_BOUNDING_BOX);


                // try to get globe coordinates from track
                vector<double> koord;

                if (calc_globe_coord)
                {
                        for (CvTracks::iterator it = tracks.begin(); it != tracks.end(); it
++)
                        {
                                if (it->second->inactive < 1)
                                {
                                passpunkt ptest;
                                ptest.pic_x = it->second->centroid.x;
                                ptest.pic_y = it->second->centroid.y;

                                koord = refgrid.calc_lat_long(ptest);
                                }
                                else
                                {
                                        old_lat = 999;
                                        old_long = 999;
```

```
                                        }


                                }
                        }

                        if (use_rotation)
                        {

                                //rotation based on quaternion

                                if ((koord.size() >= 2) && (old_lat != 999) && (old_long != 999))
                                {
                                        //round coordinates

                                        old_lat = cutDecimal(old_lat,1);
                                        old_long = cutDecimal(old_long,1);
                                        koord[0] = cutDecimal(koord[0],1);
                                        koord[1] = cutDecimal(koord[1],1);


                                        // check whether touch points are identical
                                        if (!(( old_lat == koord[0] ) &&
                                           ( old_long == koord[1])))
                                        {


                                                // insert log entry
                                            //
                                                //std::string log_entry;
                                                //std::ostringstream temp;
                                                //temp << old_lat << "," << old_long << "," <<
koord[0] << "," << koord[1] << ",";
                                                //temp << winkel << "," << drehachse.x << "," <<
drehachse.y << "," << drehachse.z;
                                                //log_entry = temp.str();
                                                //write_to_logfile
("beta_quaternion_log.csv",log_entry);


                                        //calculate quaternion
                                        //quaternion q = calculate_rot_quaternion((double)old_lat,
(double)old_long, (double) koord[0], (double) koord[1]);
                                        //quaternion q =
calculate_rot_quaternion_2((double)old_lat, (double)old_long, (double) koord[0], (double) koord[1],
old_rot);
                                        quaternion q = rot2((double)old_lat, (double)old_long,
(double) koord[0], (double) koord[1],old_rot);

                                        // check for division with 0

                                        // no -1.#INDF
                                        if (!(( q.w != q.w ) ||
                                                (q.x != q.x) ||
                                                (q.y != q.y) ||
                                                (q.z != q.z)))
                                                {

                                                        std::ostringstream strs;




                                                        // Add rotation to actual local
coordinate-system of globe
                                                        strs << "StoryAddQuaternion "  << q.w << "
" << q.x << " " << q.y << " " << q.z;


                                                        old_rot = multiply_quaternion(old_rot,q);

                                                        //old_rot.x = q.x;
                                                        //old_rot.y = q.y;
                                                        //old_rot.z = q.z;
```

```
                                                        //DEBUG
                                                        //double betrag =
sqrt(pow(q.w,2)+pow(q.x,2)+pow(q.y,2)+pow(q.z,2) );
                                                        //cout << fixed;
                                                        //cout << setprecision (6) << q.w << " | "
<< q.x  << " | " << q.y << " | " << q.z << " | " << betrag << endl;

                                                        std::string message = strs.str();
                                                        long rc = sendto(s, message.c_str(),
strlen(message.c_str()), 0, (SOCKADDR*)&Player_addr, sizeof(SOCKADDR_IN));
                                                }

                                        }
                                }
                                else
                                {
                                        //cout << "no rotation -> koord.size() < 2" << endl;
                                }



                        }

                        if (koord.size() >= 2)
                        {
                        old_lat = koord.at(0);
                        old_long = koord.at(1);
                        }

                        // Render detected Blobs
                        //cvRenderBlobs(labelImg, blobs, frame_col, frame_col);

                        // draw lines
                        if (use_lines)
                        {


                                cvLine(  frame_col,
                                                cvPoint(lines[0].x1,lines[0].y1),
                                                cvPoint(lines[0].x2,lines[0].y2),
                                                cvScalar(240,240,255,0),
                                                1,
                                                CV_AA,
                                                0);



cvLine(frame_col,cvPoint(lines[1].x1,lines[1].y1),cvPoint(lines[1].x2,lines[1].y2),cvScalar(200,255,
200,0),1,CV_AA,0);
                        }

                        // averaged kalib circle
                                cvCircle(       frame_col,
                                                        cvPoint(centre.x,centre.y),
                                                        (int) centre.radius,
                                                        cvScalar(0,0,0,0),
                                                        1,
                                                        8,
                                                        0);

                        if (draw_kreis)
                        {
                                //printf ("will draw circle: %f|%f with radius: %f\n",
kreis.mittelpunkt.x,kreis.mittelpunkt.y,kreis.radius);
                                cvCircle(       frame_col,

        cvPoint(kreis.mittelpunkt.x,kreis.mittelpunkt.y),
                                                        (int) kreis.radius,
                                                        cvScalar(0,255,255,0),
                                                        1,
                                                        8,
                                                        0);
```

```
                cvLine(         frame_col,
                                        cvPoint((int)p1.x, (int)p1.y),
                                        cvPoint((int)p2.x, (int)p2.y),
                                        cvScalar(0,255,255,0),
                                        1,
                                        8,
                                        0);

                cvLine(         frame_col,
                                        cvPoint((int)p2.x, (int)p2.y),
                                        cvPoint((int)p3.x, (int)p3.y),
                                        cvScalar(255,0,255,0),
                                        1,
                                        8,
                                        0);
                cvLine(         frame_col,
                                        cvPoint((int)p3.x, (int)p3.y),
                                        cvPoint((int)p1.x, (int)p1.y),
                                        cvScalar(255,255,0,0),
                                        1,
                                        8,
                                        0);
        }

        // Show image
        cvShowImage("blob",frame_col);

        // Release images to avoid buffer overflow
        cvReleaseImage(&frame_col);
        cvReleaseImage(&labelImg);
        //=====REMEMEBER   cvReleaseBlobs(blobs);
        //cvReleaseImageData(&frame->imageData);
}



//display frames
if (update_vid) cvShowImage("vid1",frame_bw1);
if (mask) cvShowImage("kalib",frame_kalib);



//wait for key and handle inputs
char k = cvWaitKey(1);
switch (k)
{
        case 'r':
                {
                        //record reference point

                        // read blob
                        get_blobs (ref_tracks,frame_bw1, minblobsize, maxblobsize);

                        if ((ref_tracks.size() != 1))
                        {
                                cout << "not one blob" << endl;
                                break;
                        }

                        // set focus to console
                        hwndConsole = FindWindowA( NULL, "console" );
                        SetFocus(hwndConsole);
                        SetActiveWindow(hwndConsole);
                        SetForegroundWindow(hwndConsole);


                        // read lat/lon values
                        double lat = 0;
                        double _long = 0;
                        cout << "longitude|laenge:" << endl;
                        cin >> _long;
                        cout << "latitude|breite:" << endl;
                        cin >> lat;
```

```
                                        if ((lat == 999) || (_long == 999))
                                        {
                                                break;
                                        }

                                        // check whether lat/lon pair exists already
                                        bool abort = false;
                                        for (vector<Referenzpunkt>::iterator it = ref_list.begin();
it != ref_list.end(); it++)
                                        {
                                                if (( it->latitude == lat) && ( it->longitude ==
_long))
                                                {
                                                        abort = true;
                                                        cout << "rp bereits vorhanden" << endl;
                                                }
                                        }
                                        if (abort) continue;




                                        if ((ref_tracks.size() == 1))
                                        {
                                                /*
                                                cout << "entering ref_track loop" << endl;
                                                Referenzpunkt rp;
                                                rp.x_pic = ref_tracks[1]->centroid.x;
                                                rp.y_pic = ref_tracks[1]->centroid.y;
                                                rp.latitude = lat;
                                                rp.longitude = _long;
                                                ref_list.push_back(rp);
                                                Beep(880,350);
                                                Sleep(100);
                                                Beep(880,350);
                                                */
                                                for ( CvTracks::iterator it = ref_tracks.begin();
it != ref_tracks.end(); it++)
                                                {
                                                        Referenzpunkt rp;
                                                        rp.x_pic = it->second->centroid.x;
                                                        rp.y_pic = it->second->centroid.y;
                                                        rp.latitude = lat;
                                                        rp.longitude = _long;
                                                        ref_list.push_back(rp);
                                                        Beep(880,300);
                                                        Sleep(300);
                                                        Beep(880,300);
                                                        Sleep(1000);
                                                }
                                        }
                                        else
                                        {
                                                cout << "not exactly 1 blob" << endl;
                                        }/*
                                        if (blobs.size() == 1)
                                        {
                                                for (CvBlobs::const_iterator it=blobs.begin(); it!
=blobs.end(); ++it)
                                                {
                                                        Referenzpunkt rp;
                                                        rp.x_pic = it->second->centroid.x;
                                                        rp.y_pic = it->second->centroid.y;
                                                        rp.latitude = lat;
                                                        rp.longitude = _long;
                                                        ref_list.push_back(rp);
                                                        Beep(880,350);
                                                        Sleep(100);
                                                        Beep(880,350);

                                                }
                                        }*/
                                        //cvReleaseBlobs(blobs);
```

```
                                        //set focus to video window
                                        // set focus to console
                                        hwndConsole = FindWindowA( NULL, "vid1" );
                                        SetFocus(hwndConsole);
                                        SetActiveWindow(hwndConsole);
                                        SetForegroundWindow(hwndConsole);


                                        break;
                                }
                        case 'd':
                                use_blob = !use_blob;
                                if (use_blob)
                                {
                                        printf ("Blob detection is ON\n");
                                }
                                else
                                {
                                        printf ("Blob detection is OFF\n");
                                }
                                break;
                        case 'q':
                                quit = true;
                                break;
                        case 'm':
                                mask = !mask;
                                if (!mask)
                                {
                                        //copy kalib to overlay
                                        cvCopy(frame_kalib,mp_overlay);
                                        //put text in overlay

        cvRectangle(mp_overlay,cvPoint(7,105),cvPoint(140,140),cvScalar(0,0,0,0),CV_FILLED);
                                        cvPutText(mp_overlay, "Stopped", cvPoint(10, 130), &font,
cvScalar(200, 200, 200, 0));
                                        //blend with kalib_image
                                        //cvAdd(mp_overlay,frame_kalib,mp_overlay);

                                        //showImage
                                        cvShowImage("kalib", mp_overlay);
                                }
                                break;
                        case 'k':
                                use_kalib = !use_kalib;
                                if (use_kalib)
                                {
                                        printf ("useage of kalibration image is ON\n");
                                }
                                else
                                {
                                        printf ("useage of kalibration image is OFF\n");
                                }
                                break;
                        case 'p':
                                pause = !pause;
                                break;
                        case '+':
                                gaus += 2;
                                printf ("gaus value = %d\n",gaus);
                                break;
                        case '-':
                                if (gaus > 1)
                                {
                                        gaus -= 2;
                                }
                                printf ("gaus value = %d\n",gaus);
                                break;
                        case 's':
                                use_smooth = !use_smooth;
                                if (use_smooth)
                                {
                                        printf ("useage of smoothing image is ON\n");
                                }
                                else
                                {
```

```
                                printf ("useage of smoothing image is OFF\n");
                        }
                        break;
                case 'v':
                        update_vid = !update_vid;
                        if (update_vid)
                        {
                                printf ("video update is ON\n");
                        }
                        else
                        {
                                printf ("video update is OFF\n");
                        }
                        break;
                case 'e':
                        use_equal = !use_equal;
                        if (use_equal)
                        {
                                printf ("useage of histequalize is ON\n");
                        }
                        else
                        {
                                printf ("useage of histequalize is OFF\n");
                        }
                        break;
                case 'c':

                        //calculate refgrid
                        {
                                cout << "calculating refgrid" << endl;
                                sort = sort_ref_points(ref_list);
                                refgrid.ref_points = &ref_list;
                                refgrid.sort = &sort;
                                mPunkt centerpoint = get_center_from_refpoints(ref_list);
                                refgrid.center.x = centerpoint.x;
                                refgrid.center.y = centerpoint.y;
                                for (int i=0; i < refgrid.ref_points->size(); i++)
                                {
                                        // calculate r and phi for every refpoint
                                        //polarkoordinate kart2polar (double zentrumx,
double zentrumy, double x, double y)
                                        polarkoordinate pktemp = kart2polar (centerpoint.x,
centerpoint.y, refgrid.ref_points->at(i).x_pic, refgrid.ref_points->at(i).y_pic);
                                        refgrid.ref_points->at(i).r = pktemp.r;
                                        refgrid.ref_points->at(i).phi = pktemp.phi;
                                }

                                //Setting roi
                                for (vector<Referenzpunkt>::iterator it = ref_list.begin();
it!=ref_list.end(); it++)
                                {
                                        if (it->x_pic < xmin) xmin = it->x_pic;
                                        if (it->x_pic > xmax) xmax = it->x_pic;
                                        if (it->y_pic < ymin) ymin = it->y_pic;
                                        if (it->y_pic > ymax) ymax = it->y_pic;
                                }
                                roi_change = true;


                        }
                        calc_globe_coord = true;
                        break;

                case 't':
                        use_vthres = !use_vthres;
                        if (use_vthres)
                        {
                                printf ("useage of vthresh is ON\n");
                        }
                        else
                        {
                                printf ("useage of vthresh is OFF\n");
                        }
                        break;
                case 'a':
```

```
                                {
                                        // einen kreis aufnehmen
                                        double x = 0;
                                        double y = 0;

                                        if (blobs.size() != 1)
                                        {
                                                printf ("there is less or more then 1 blob....\n");
                                                break;
                                        }

                                        if (p_counter == 3)
                                        {
                                                p_counter = 0;
                                                draw_kreis = false;
                                        }


                                        for (CvBlobs::const_iterator it=blobs.begin(); it!
=blobs.end(); ++it)
                                        {
                                          //cout << "Blob #" << it->second->label << ": Area=" <<
it->second->area << ", Centroid=(" << it->second->centroid.x << ", " << it->second->centroid.y <<
")" << endl;
                                          //x =  it->second->centroid.x;
                                          //y =  it->second->centroid.y;
                                          x = (it->second->maxx + it->second->minx)/2;
                                          y = (it->second->maxy + it->second->miny)/2;
                                        }
                                        cvReleaseBlobs(blobs);

                                        switch (p_counter)
                                        {
                                        case 0:
                                                {
                                                        p1.x = x;
                                                        p1.y = y;
                                                        p_counter += 1;
                                                        //printf ("p1 set; %f | %f\n", p1.x,p1.y);
                                                        printf("1");
                                                        Beep(640,100);
                                                        break;
                                                }
                                        case 1:
                                                {
                                                        p2.x = x;
                                                        p2.y = y;
                                                        p_counter += 1;
                                                        //printf ("p2 set; %f | %f\n", p2.x,p2.y);
                                                        printf ("2");
                                                        Beep(640,100);Beep(640,100);
                                                        break;
                                                }
                                        case 2:
                                                {
                                                        p3.x = x;
                                                        p3.y = y;
                                                        p_counter += 1;
                                                        //printf ("p3 set; %f | %f\n", p3.x,p3.y);
                                                        kreis = _3punktkreis(p1,p2,p3);
                                                        draw_kreis = true;
                                                        printf ("3\n");
                                                        Beep(640,100);Beep(640,100);Beep(640,100);
                                                        printf ("centre:  %.1f|%.1f\tradius:
%.1f\n",kreis.mittelpunkt.x,kreis.mittelpunkt.y,kreis.radius);

                                                        centre.nr += 1;
                                                        centre.xsum += kreis.mittelpunkt.x;
                                                        centre.ysum += kreis.mittelpunkt.y;
                                                        centre.radsum += kreis.radius;
                                                        centre.calculate();
                                                        centre.print();


                                                        break;
```

```
                                                }
                                        }

                                        break;
                                }
                        case 'b':
                                {
                                        // rotation benutzen
                                        use_rotation = !use_rotation;
                                        if (use_rotation)
                                        {
                                                printf ("useage of rotation is ON\n");
                                        }
                                        else
                                        {
                                                printf ("useage of rotation is OFF\n");
                                        }
                                        break;
                                }
                        /*case 'l':
                                {
                                        use_lines = !use_lines;
                                        if (use_lines)
                                        {
                                                printf ("useage of lines is ON\n");
                                        }
                                        else
                                        {
                                                printf ("useage of lines is OFF\n");
                                        }
                                        break;
                                }*/
                        case 'w':
                                {
                                        // export ref points to a file
                                        write_ref_points_to_file("refpoints.txt",ref_list);
                                        break;
                                }
                        case 'l':
                                {
                                        //load ref_points from file
                                        std::string filename = "refpoints.txt";
                                        ref_list.clear();
                                        fill_ref_list(filename,ref_list);
                                        break;
                                }
                        case 'x':
                                {
                                        std::ostringstream strs;
                                        quaternion q  = calculate_rot_quaternion(0,0,0,-1);
                                        strs << "StoryAddQuaternion "  << q.w << " " << q.x << " "
<< q.y << " " << q.z;
                                        std::string message = strs.str();
                                        long rc = sendto(s, message.c_str(),
strlen(message.c_str()), 0, (SOCKADDR*)&Player_addr, sizeof(SOCKADDR_IN));
                                        break;
                                }
                        case 'X':
                                {
                                        std::ostringstream strs;
                                        quaternion q  = calculate_rot_quaternion(0,0,0,1);
                                        strs << "StoryAddQuaternion "  << q.w << " " << q.x << " "
<< q.y << " " << q.z;
                                        std::string message = strs.str();
                                        long rc = sendto(s, message.c_str(),
strlen(message.c_str()), 0, (SOCKADDR*)&Player_addr, sizeof(SOCKADDR_IN));
                                        break;
                                }
                        case 'y':
                                {
                                        std::ostringstream strs;
                                        quaternion q  = calculate_rot_quaternion(0,1,1,0);
                                        strs << "StoryAddQuaternion "  << q.w << " " << q.x << " "
<< q.y << " " << q.z;
                                        std::string message = strs.str();
```

```
                                        long rc = sendto(s, message.c_str(), 
strlen(message.c_str()), 0, (SOCKADDR*)&Player_addr, sizeof(SOCKADDR_IN));
                                        break;
                                }
                        case 'Y':
                                {
                                        std::ostringstream strs;
                                        quaternion q  = calculate_rot_quaternion(1,0,0,1);
                                        strs << "StoryAddQuaternion "  << q.w << " " << q.x << " " 
<< q.y << " " << q.z;
                                        std::string message = strs.str();
                                        long rc = sendto(s, message.c_str(), 
strlen(message.c_str()), 0, (SOCKADDR*)&Player_addr, sizeof(SOCKADDR_IN));
                                        break;
                                }
                        case 'z':
                                {
                                        std::ostringstream strs;
                                        quaternion q  = calculate_rot_quaternion(0,0,-1,0);
                                        strs << "StoryAddQuaternion "  << q.w << " " << q.x << " " 
<< q.y << " " << q.z;
                                        std::string message = strs.str();
                                        long rc = sendto(s, message.c_str(), 
strlen(message.c_str()), 0, (SOCKADDR*)&Player_addr, sizeof(SOCKADDR_IN));
                                        break;
                                }
                        case 'Z':
                                {
                                        std::ostringstream strs;
                                        quaternion q  = calculate_rot_quaternion(0,0,1,0);
                                        strs << "StoryAddQuaternion "  << q.w << " " << q.x << " " 
<< q.y << " " << q.z;
                                        std::string message = strs.str();
                                        long rc = sendto(s, message.c_str(), 
strlen(message.c_str()), 0, (SOCKADDR*)&Player_addr, sizeof(SOCKADDR_IN));
                                        break;
                                }
                        case 'u':
                                {
                                        // DEBUG -> STORY auswaehen
                                        std::ostringstream strs2;
                                        strs2 << "StoryOpen touchtest1";
                                        std::string message2 = strs2.str();
                                        rc = sendto(s, message2.c_str(), strlen(message2.c_str()), 
0, (SOCKADDR*)&Player_addr, sizeof(SOCKADDR_IN));
                                        std::ostringstream strs3;
                                        strs3 << "StoryPause";
                                        message2 = strs3.str();
                                        rc = sendto(s, message2.c_str(), strlen(message2.c_str()), 
0, (SOCKADDR*)&Player_addr, sizeof(SOCKADDR_IN));
                                        old_rot.w = 1;
                                        old_rot.x = 0;
                                        old_rot.y = 0;
                                        old_rot.z = 0;
                                        break;
                                }


                }

                total_frames += 1;
        }


        // End Image Processing Loop <-
        //===============================


        //##########################################################################################
        //===============================
        // -> Clean up

        // STOP THE CAMERA
        FiStopCamera(g_hCamera);
    FiTerminate();
```

```
        cvDestroyAllWindows();
        printf ("number of missed frames: %d",missed_frames);
        //Sleep(3000);
        system("PAUSE");
        return 0;
}
```

QUELLTEXT: BLOB_FUNCTIONS.H

```
#pragma once
#include <cvblob.h>

#ifndef MYUBCORE
#define MYUBCORE
#include "my_ubcore.h"
#endif



using namespace cvb;


void get_blobs (CvTracks &tracks, IplImage* in_image, int minblobsize, int maxblobsize);
```

QUELLTEXT: BLOB_FUNCTIONS.CPP

```
#pragma once
#include "blob_functions.h"

void get_blobs (CvTracks &tracks, IplImage* in_image, int minblobsize, int maxblobsize)
{
	unsigned int result;


	CvBlobs blobs;
	IplImage* labelimg = cvCreateImage(cvSize(800,600), IPL_DEPTH_LABEL, 1);
	// analyze image and get blobs
	result=cvLabel(in_image, labelimg, blobs);


	// Filter Blobs
	//cvFilterByArea(blobs, minblobsize, 10 * maxblobsize);
	cvFilterByArea(blobs, minblobsize, 10 * maxblobsize);

	//Tracks
	cvUpdateTracks(blobs, tracks, 100., 2);
	cvReleaseBlobs(blobs);
	cvReleaseImage(&labelimg);
}
```

QUELLTEXT: KALIB_FUNCTIONS_AND_TYPES.H

```
#pragma once

#ifndef KALIB_DEF
#define KALIB_DEF


#define _USE_MATH_DEFINES
#include <math.h>
#include <vector>
#include <iostream>
#include <string>
#include <list>
#include <fstream>
#include "Referenzpunkt.h"
#include "kreis_berechnungen.h"

#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>

#include <opencv2\opencv.hpp>
#include <opencv\highgui.h>
#include <opencv\cxcore.h>
#include <opencv\cv.h>

using namespace std;



#define PP_THRESHOLD     0x0001
#define PP_EQUALIZE              0x0002
#define PP_SMOOTH                0x0010
#define PP_KALIB         0x0020




struct passpunkt
{
        double pic_x;
        double pic_y;
        double globe_lat;
        double globe_long;

};



struct polarkoordinate
{
        double r;
        double phi;
};


struct koordinate
{
        double x;
        double y;
};


struct _3tuple
{
        int x;
        int y;
        int z;
};
```

```
struct latlist
{
        double lat;
        vector<Referenzpunkt*> ref_points;
};

struct quaternion
{
        double w;
        double x;
        double y;
        double z;
};

struct longlist
{
        double _long;
        vector<Referenzpunkt*> ref_points;
};

struct sorted_ll_list
{
        vector<latlist> lats;
        vector<longlist> longs;
};

struct xyz_vektor
{
        double x;
        double y;
        double z;
};

struct euler_rotation
{
        double yaw;
        double pitch;
        double roll;
};


struct matrix_3x3
{
        double a1;
        double a2;
        double a3;
        double b1;
        double b2;
        double b3;
        double c1;
        double c2;
        double c3;
};


passpunkt new_pp(void);

//Umrechnung Karthesisches -> Polarsystem
polarkoordinate kart2polar (double zentrumx, double zentrumy, double x, double y);


//Umrechnung Polar -> Karthesisches System
koordinate polar2kart (double zentrumx, double zentrumy, double r, double phi);


// Alle k-Kombinationen aus einer n-Liste
void subset(    int arr[],
                                int size,
                                int left,
                                int index,
                                vector<int> &l,
                                vector<_3tuple> &out);
```

```
// Einen String zwischen Seperatorzeichen aufsplitten

vector<string> splitstring (string in, char sep);

template <class T>
vector<_3tuple> get_3kombis_from_vector (T in_vec);


// zeilenweises einlesen einer datei
vector<string> read_lines (string filename);


// fill ref_list from std pattern input file (x,y,lon,lat)
// dies funktion ist nur zum debuggen ohne globus gedacht,
// für den fall das refpunkte aus einer datei ausgelesen werden
void fill_ref_list (string filename, vector<Referenzpunkt> &ref_list);


// sortiert eine liste von referenzpunkten so,
// dass zeiger auf alle refpunkte mit gleicher länge bzw breite in einem vektor gesammelt werden
sorted_ll_list sort_ref_points (vector<Referenzpunkt> &ref_list);

// funktion zum errechnen eines gemittelten kreis-mittelpunkts
// aus allen ref punkten die die selbe latitude haben alle 3er kombis bestimmen
// für jede 3er kombi den mittelpunkt ausrechnen und sammeln
// mittelwert bestimmen
mPunkt get_center_from_refpoints (vector<Referenzpunkt> &ref_list);

// Ausgabe der referenzpunkte in eine datei
void write_ref_points_to_file (std::string filename, vector<Referenzpunkt> &ref_list);

// Eintrag in ein Logfile schreiben
void write_to_logfile (std::string filename, std::string content);


//Eine Zahl auf n Nachkommastelen kuerzen
float cutDecimal( float myNumber, int decimal);

void process_image(IplImage *image, unsigned short flags);

// Umrechnung von geogrpahischen Kugelkoordinaten in einen (x,y,z)Vektor an der Einheitskugel
xyz_vektor lat_long_2_vek (float lat, float lon);
xyz_vektor lat_long_2_vek_2 (double lat, double lon);

// Kreuzprodukt aus zwei Vektoren = Normalenvektor einer Ebene
xyz_vektor normalenvektor (xyz_vektor u, xyz_vektor v);


// Einheitsquaternion für die Drehung von Punkt1(länge,breite) -> Punkt2(länge,breite) berechnen
quaternion calculate_rot_quaternion ( double lat1, double lon1, double lat2, double lon2);

// Einheitsquaternion für die Drehung von Punkt1(länge,breite) -> Punkt2(länge,breite) berechnen
// + Anpassung der Drehachse an das aktuelle Rotationsquaternion (qr)
quaternion calculate_rot_quaternion_2 ( double lat1, double lon1, double lat2, double lon2,
quaternion qr);

// Winkel zwischen zwei Vektoren
double vektoren_winkel (xyz_vektor u, xyz_vektor v);

// Degree 2 Radian
double rad(double input);

// rad 2 deg
double deg(double input);

// Ein Quaternion normalisieren "Einheitsquaternion
quaternion normalize_quaternion (double w, double x, double y, double z);

// Umwandlung eines Quaternions in Eulersche Winkel (yaw, pitch, roll)
euler_rotation quaternion_2_euler(quaternion q);

// Multiplikation zweier Quaternions
```

```
quaternion multiply_quaternion (quaternion q1, quaternion q2);

// Quaternion invertieren
quaternion invert_quaternion (quaternion q);

// Rotations_Quaternion in eine 3x3 Matrix umrechnen
matrix_3x3 quat_2_matr (quaternion q);

// Vektor mir einer 3x3 Matrix multiplizieren
xyz_vektor vek_x_matr (xyz_vektor v, matrix_3x3 m);

// Rotation zur alten hinzufügen
quaternion rot2 (double lat1, double lon1, double lat2, double lon2, quaternion qr);

#endif
```

```
#pragma once

#include "kalib_functions_and_types.h"

/*#define _USE_MATH_DEFINES
#include <math.h>
#include <vector>
#include <iostream>
#include <string>
#include <list>
#include <fstream>
#include "Referenzpunkt.h"
#include "kreis_berechnungen.h"*/

using namespace std;


passpunkt new_pp()
{
        passpunkt pp;
        pp.pic_x = 0;
        pp.pic_y = 0;
        pp.globe_long = 0;
        pp.globe_lat = 0;
        return pp;
}

//Umrechnung Karthesisches -> Polarsystem
polarkoordinate kart2polar (double zentrumx, double zentrumy, double x, double y)
{
        double r = sqrt(pow(x-zentrumx,2)+pow(y-zentrumy,2));
        double phi = atan2(y-zentrumy,x-zentrumx);
        phi = phi * (180/M_PI);
        polarkoordinate pk;

        // phi in 0-360 umwandeln
        // atan2 =>
        // sektor I   : -90 -> -0
        // sektor II  : +0 -> +90
        // sektor III : +90 -> + 180
        //sektor  IV  : -180 -> - 90

        if ((phi > -180) && (phi < 0))   phi = 360 + phi;
        if (phi == -180) phi = 180;
        if (phi == 360) phi = 0;



        pk.phi = phi;
        pk.r = r;
        return pk;
}

//Umrechnung Polar -> Karthesisches System
koordinate polar2kart (double zentrumx, double zentrumy, double r, double phi)
{
        phi = phi * (M_PI/180);
        double x = r * cos(phi) + zentrumx;
        double y = r * sin(phi) + zentrumy;
        koordinate k = {x,y};
        return k;
}


// Alle k-Kombinationen aus einer n-Liste
void subset(    int arr[],
                                int size,
```

```
                                int left,
                                int index,
                                vector<int> &l,
                                vector<_3tuple> &out)
{
    if(left==0){
                _3tuple tmp = {0,0,0};
                int tmp2[3] = {0,0,0};
                int i = 0;
                for(vector<int>::iterator it=l.begin(); it!=l.end() ; ++it)
                {
                        tmp2[i] = *it;
                        i += 1;
                }
                tmp.x = tmp2[0];
                tmp.y = tmp2[1];
                tmp.z = tmp2[2];
                out.push_back(tmp);
        return;
    }
    for(int i=index; i<size;i++){
        l.push_back(arr[i]);
        subset(arr,size,left-1,i+1,l,out);
        l.pop_back();
    }

}




vector<string> splitstring (string in, char sep)
{
        vector<string> output;
        vector<int> seps;


        for (unsigned int i = 0; i < in.size(); i++)
        {
                if (in[i] == sep)
                {
                        seps.push_back(i);
                }
        }

        seps.push_back(in.size());
        seps.insert(seps.begin(),-1);


        for (unsigned int i=0 ; i < seps.size() -1; i++)
        {
                output.push_back(in.substr(seps[i]+1,seps[i+1] - (seps[i] +1)));
        }

        return output;

}


template <class T>
vector<_3tuple> get_3kombis_from_vector (T in_vec)
{
        //generate int array
        int* int_array = NULL;
        int_array = new int[in_vec.size()];

        //parameters to subset function
        vector<int> lt;
        vector<_3tuple> kombis;

        koordinate m_punkt = {0,0}; // mittelpunkt


        for (unsigned int i=0; i<in_vec.size();i++)
        {
```

```
		int_array[i] = i;
	}

	subset(		int_array,
				in_vec.size(),
				3,
				0,
				lt,
				kombis);

	return kombis;
}


// zeilenweises einlesen einer datei
vector<string> read_lines (string filename)
{
	vector<string> lines;
	string line;

	ifstream myfile (filename);
	while (getline(myfile, line))
	{
		lines.push_back(line);
	}

	myfile.close();
	return lines;
}


// fill ref_list from std pattern input file (x,y,lon,lat)
// dies funktion ist nur zum debuggen ohne globus gedacht,
// für den fall das refpunkte aus einer datei ausgelesen werden
void fill_ref_list (string filename, vector<Referenzpunkt> &ref_list)
{
	vector<string> lines = read_lines(filename);

	for (unsigned int i = 0; i < lines.size(); i++)
	{
		vector<string> items = splitstring(lines[i],',');

		Referenzpunkt rp;
		rp.x_pic = atoi (items[0].c_str());
		rp.y_pic = atoi (items[1].c_str());
		rp.latitude = atoi (items[3].c_str());
		rp.longitude = atoi (items[2].c_str());

		ref_list.push_back(rp);
	}

}


// sortiert eine liste von referenzpunkten so,
// dass zeiger auf alle refpunkte mit gleicher länge bzw breite in einem vektor gesammelt werden
sorted_ll_list sort_ref_points (vector<Referenzpunkt> &ref_list)
{
	vector <latlist> ll;
	vector <longlist> lo;
	vector<double> lats;
	vector<double> longs;
	sorted_ll_list sorll;

	lats.push_back(ref_list[0].latitude);

	for (unsigned int i = 1; i<ref_list.size();i++)
	{
		bool exist = false;
		for (unsigned int j=0; j<lats.size();j++)
		{
			if (lats[j] == ref_list[i].latitude) exist = true;

		}
		if (!exist) lats.push_back(ref_list[i].latitude);
```

```
	}

	for (unsigned int i=0; i<lats.size(); i++)
	{
		latlist l;
		l.lat = lats[i];
		for (unsigned int j=0; j<ref_list.size();j++)
		{
			if (lats[i] == ref_list[j].latitude) l.ref_points.push_back(&ref_list[j]);
		}
		ll.push_back(l);
	}

	sorll.lats = ll;

	longs.push_back(ref_list[0].longitude);

	for (unsigned int i = 1; i<ref_list.size();i++)
	{
		bool exist = false;
		for (unsigned int j=0; j<longs.size();j++)
		{
			if (longs[j] == ref_list[i].longitude) exist = true;
		}
		if (!exist) longs.push_back(ref_list[i].longitude);
	}

	for (unsigned int i=0; i<longs.size(); i++)
	{
		longlist l;
		l._long = longs[i];
		for (unsigned int j=0; j<ref_list.size();j++)
		{
			if (longs[i] == ref_list[j].longitude)
l.ref_points.push_back(&ref_list[j]);
		}
		lo.push_back(l);
	}

	sorll.longs = lo;

	return sorll;
}


// funktion zum errechnen eines gemittelten kreis-mittelpunkts
// aus allen ref punkten die die selbe latitude haben alle 3er kombis bestimmen
// für jede 3er kombi den mittelpunkt ausrechnen und sammeln
// mittelwert bestimmen
mPunkt get_center_from_refpoints (vector<Referenzpunkt> &ref_list)
{
	zentrum center;
	sorted_ll_list sort = sort_ref_points (ref_list);

	for (unsigned int i=0; i<sort.lats.size(); i++)
	{	//loop durch alle breiten


		// 3er kombis aus allen punkten der selben breite
		vector<_3tuple> kombis = get_3kombis_from_vector (sort.lats[i].ref_points);

		for (unsigned int k=0; k<kombis.size(); k++)
			{	//loop durch alle 3er-kombis

				mPunkt p1 = {0,0};
				p1.x = sort.lats[i].ref_points[kombis[k].x]->x_pic;
				p1.y = sort.lats[i].ref_points[kombis[k].x]->y_pic;

				mPunkt p2 = {0,0};
				p2.x = sort.lats[i].ref_points[kombis[k].y]->x_pic;
				p2.y = sort.lats[i].ref_points[kombis[k].y]->y_pic;

				mPunkt p3 = {0,0};
				p3.x = sort.lats[i].ref_points[kombis[k].z]->x_pic;
				p3.y = sort.lats[i].ref_points[kombis[k].z]->y_pic;
```

```
                                mKreis kreis =  _3punktkreis (p1,p2,p3);

                                // Prüfe ob eine Division durch 0 stattfand, ignoriere Ergebnis
wenn ja
                                if ((kreis.mittelpunkt.x == kreis.mittelpunkt.x) &&
(kreis.mittelpunkt.y == kreis.mittelpunkt.y))
                                {
                                center.xsum += kreis.mittelpunkt.x;
                                center.ysum += kreis.mittelpunkt.y;
                                center.nr += 1;

                                }

                                /*
                                cout << p1.x << "|" << p1.y << "\t";
                                cout << p2.x << "|" << p2.y << "\t";
                                cout << p3.x << "|" << p3.y << "\t-->";
                                cout << kreis.mittelpunkt.x << " | " << kreis.mittelpunkt.y <<
endl;
                                cout <<
"----------------------------------------------------------\n";
                                */

                        }


        }
        mPunkt rv;
        rv.x = center.xsum / center.nr;
        rv.y =  center.ysum / center.nr;
        //cout << "zentrum: " << center.xsum / center.nr << " | " << center.ysum / center.nr <<
endl;
        return rv;
}


/*
#define PP_THRESHOLD    0x0001
#define PP_EQUALIZE             0x0002
#define PP_SMOOTH               0x0010
#define PP_KALIB        0x0020
*/


// preprocessing für ein bild
// gesteuert über flags
void process_image (IplImage *image, unsigned short flags)
{
 if (flags&PP_THRESHOLD) cout << "threshold" << endl;
 if (flags&PP_SMOOTH)    cout << "smooth" << endl;
}



void write_ref_points_to_file (std::string filename, vector<Referenzpunkt> &ref_list)
{
        ofstream myfile;
        myfile.open (filename, ios::trunc);

        for (vector<Referenzpunkt>::iterator it = ref_list.begin(); it != ref_list.end(); it++)
        {
                myfile << it->x_pic << "," << it->y_pic << "," << it->longitude << "," << it-
>latitude << endl;
        }

        myfile.close();

}

void write_to_logfile (std::string filename, std::string content)
{
        ofstream myfile;
        myfile.open(filename, ios::app);
```

```
        myfile << content << endl;
        myfile.close();
}

float cutDecimal( float myNumber, int decimal){

    myNumber = myNumber*( pow(float(10),decimal) );
    myNumber = (int)myNumber;
    myNumber = myNumber/( pow(float(10),decimal) ) ;
    return myNumber;
}


// Degree 2 Radian
double rad(double input)
{
        //grad -> bogenmass
        //float ret_val = cos_winkel * (3.1415926535/180);
        return input / (180/M_PI);

}

double deg(double input)
{
        return input * (180/M_PI);
}

// Umrechnung von geographischen Kugelkoordinaten in einen (x,y,z)Vektor an der Einheitskugel
xyz_vektor lat_long_2_vek_2 (double lat, double lon)
{
        if (lon > 180) lon = lon - 360;

        //orginal
        double x = cos(rad(lat))*cos(rad(lon));
        double y = cos(rad(lat))*sin(rad(lon));
        double z = sin(rad(lat));

        // verdrehte c4d/ogre achsen?
        //float x = cos(rad(lat))*cos(rad(lon));
        //float z = -1 * (cos(rad(lat))*sin(rad(lon)));
        //float y = -1 * (sin(rad(lat)));


        xyz_vektor ret_val;
        ret_val.x = x;
        ret_val.y = y;
        ret_val.z = z;

        return ret_val;
}
xyz_vektor lat_long_2_vek (float lat, float lon)
{
        float x,y,z = 0;

        lat = lat/180*3.1415926535;
        lon = lon/180*3.1415926535;

        int wahl = 0;


        // x y z koordinaten

        if (wahl == 0)
        {
        x = sin(lat) * cos(lon);
        y = sin(lat)+sin(lon);
        z = cos(lat);
        }

        if (wahl == 1)
        {
        x = cos(lat) * cos(lon);
        y = cos(lat) * sin (lon);
        z = sin(lat);
```

```
        }

        //normieren = einheitsvektor erstellen
        /*
        float betrag = ((x*x) + (y*y) + (z*z));
        x = x / betrag;
        y = y / betrag;
        z = z / betrag;
        */

        xyz_vektor ret_val;
        ret_val.x = x;
        ret_val.y = y;
        ret_val.z = z;

        return ret_val;
}


// Ein Quaternion normalisieren "Einheitsquaternion
quaternion normalize_quaternion (double w, double x, double y, double z)
{
        double betrag = sqrt(pow((float)w,2)+pow((float)x,2)+pow((float)y,2)+pow((float)z,2));
        quaternion ret_val;
        ret_val.w = w / betrag;
        ret_val.x = x / betrag;
        ret_val.y = y / betrag;
        ret_val.z = z / betrag;

        return ret_val;
}

// Kreuzprodukt aus zwei Vektoren = Normalenvektor einer Ebene
xyz_vektor normalenvektor (xyz_vektor v1, xyz_vektor v2)
{
        xyz_vektor ret_val;
        double x,y,z,betrag = 0;

        x = (v1.y*v2.z) - (v2.y*v1.z);
        y = (v1.z*v2.x) - (v2.z*v1.x);
        z = (v1.x*v2.y) - (v2.x*v1.y);

        betrag = sqrt(pow(x,2)+pow(y,2)+pow(z,2));

        ret_val.x = x / betrag;
        ret_val.y = y / betrag;
        ret_val.z = z / betrag;

        return ret_val;
}

// Winkel zwischen zwei Vektoren
double vektoren_winkel (xyz_vektor u, xyz_vektor v)
{

        xyz_vektor v1 = u;
        xyz_vektor v2 = v;

        double cos_lambda, lambda = 0;
        cos_lambda = v1.x*v2.x + v1.y*v2.y + v1.z*v2.z;
        lambda = acos(cos_lambda);
        return lambda;
}


// Umwandlung eines Quaternions in Eulersche Winkel (yaw, pitch, roll)
euler_rotation quaternion_2_euler(quaternion q)
{
        // Umwandlung Quaternion (v.x,v.y,v.z,winkel) in euler_winkel (yaw,pitch,roll)


        float yaw = atan2(2*q.y*q.w-2*q.x*q.z , 1 - 2*(q.y*q.y) - 2*(q.z*q.z));
        float pitch = asin(2*q.x*q.y + 2*q.z*q.w);
        float roll = atan2(2*q.x*q.w-2*q.y*q.z , 1 - 2*(q.x*q.z) - 2*(q.z*q.z));
```

```
        // Sonderfaelle Nord und Suedpol


        if ((q.x*q.y + q.z*q.w) == .5)
        {
                yaw = 2*atan2(q.x,q.w);
                roll = 0;
        }

        if ((q.x*q.y + q.z*q.w) == -.5)
        {
                yaw = -2*atan2(q.x,q.w);
                roll = 0;
        }



        euler_rotation ret_val;

        //bogenmass -> grad
        ret_val.yaw = yaw * 180 /3.1415926535;
        ret_val.pitch = pitch* 180 /3.1415926535;
        ret_val.roll = roll* 180 /3.1415926535;



        return ret_val;

}

// Einheitsquaternion für die Drehung von Punkt1(länge,breite) -> Punkt2(länge,breite) berechnen
quaternion calculate_rot_quaternion ( double lat1, double lon1, double lat2, double lon2)
{
        // Variablen
        double theta1, phi1, theta2, phi2 = 0;   // Drehwinkel für die Berechnung der x,y,z
Koordinaten der Winkel
        xyz_vektor v1,v2;                                                      // Vektoren (0,0)->P1|P2
        xyz_vektor n;                                                          // Normaleneinheitsvektor
zwischen v1, v2
        double lambda = 0;                                                     // Winkel zwischen den
beiden Vektoren
        quaternion q;                                                          // zu berechnendes
Quaternion


        // Lat/Long in theta/phi umrechen
        theta1 = rad(90 - lat1);
        theta2 = rad(90 - lat2);

        phi1 = (lon1>180) ? rad(lon1 - 360) : rad(lon1);
        phi2 = (lon2>180) ? rad(lon2 - 360) : rad(lon2);


        // x,y,z Vektoren aus den Länge/Breite Punkten berechnen
        //v1.x = sin(theta1)*cos(phi1);
        //v1.y = sin(theta1)*sin(phi1);
        //v1.z = cos(theta1);
        //
        //v2.x = sin(theta2)*cos(phi2);
        //v2.y = sin(theta2)*sin(phi2);
        //v2.z = cos(theta2);

        // Anpassung an das OGRE Koordinatensystem
        // ogre koordinatensystem == tthg koordinatensystem
        // z == -x
        // x == -y
        // y == z
        v1.z = -1 * sin(theta1)*cos(phi1);
        v1.x = -1 * sin(theta1)*sin(phi1);
        v1.y = cos(theta1);

        v2.z = -1 * sin(theta2)*cos(phi2);
        v2.x = -1 * sin(theta2)*sin(phi2);
        v2.y = cos(theta2);
```

```
        // lambda bestimmen
        lambda = vektoren_winkel(v1,v2);


        // Drehachse
        n = normalenvektor(v1,v2);


        // Quaternion berechnen
        q.w = cos(lambda/2);
        q.x = sin(lambda/2)*n.x;
        q.y = sin(lambda/2)*n.y;
        q.z = sin(lambda/2)*n.z;

        //// insert log entry
        //std::string log_entry;
        //std::ostringstream temp;
        //temp << fixed;
        //temp << setprecision(6) << lat1 << "," << lon1 << "," << lat2 << "," << lon2 << ",";
        //temp << setprecision(6) <<q.w << "," << q.x << "," << q.y << "," << q.z;
        //log_entry = temp.str();
        //write_to_logfile ("beta_quaternion_log.csv",log_entry);

        return q;
}

// Einheitsquaternion für die Drehung von Punkt1(länge,breite) -> Punkt2(länge,breite) berechnen
// + Anpassung der Drehachse an das aktuelle Rotationsquaternion (qr)
quaternion calculate_rot_quaternion_2 ( double lat1, double lon1, double lat2, double lon2,
quaternion qr)
{
                // Variablen
        double theta1, phi1, theta2, phi2 = 0;   // Drehwinkel für die Berechnung der x,y,z
Koordinaten der Winkel
        xyz_vektor v1,v2;                                                   // Vektoren (0,0)->P1|P2
        xyz_vektor n;                                                       // Normaleneinheitsvektor
zwischen v1, v2
        double lambda = 0;                                                  // Winkel zwischen den
beiden Vektoren
        quaternion q;                                                       // zu berechnendes
Quaternion


        // Lat/Long in theta/phi umrechen
        theta1 = rad(90 - lat1);
        theta2 = rad(90 - lat2);

        phi1 = (lon1>180) ? rad(lon1 - 360) : rad(lon1);
        phi2 = (lon2>180) ? rad(lon2 - 360) : rad(lon2);


        // x,y,z Vektoren aus den Länge/Breite Punkten berechnen
        //v1.x = sin(theta1)*cos(phi1);
        //v1.y = sin(theta1)*sin(phi1);
        //v1.z = cos(theta1);
        //
        //v2.x = sin(theta2)*cos(phi2);
        //v2.y = sin(theta2)*sin(phi2);
        //v2.z = cos(theta2);

        // Anpassung an das OGRE Koordinatensystem
        // ogre koordinatensystem == mein koordinatensystem
        // z == -x
        // x == -y
        // y == z
        v1.z = -1 * sin(theta1)*cos(phi1);
        v1.x = -1 * sin(theta1)*sin(phi1);
        v1.y = cos(theta1);

        v2.z = -1 * sin(theta2)*cos(phi2);
        v2.x = -1 * sin(theta2)*sin(phi2);
        v2.y = cos(theta2);
```

```
        // lambda bestimmen
        lambda = vektoren_winkel(v1,v2);


        // Drehachse
        n = normalenvektor(v1,v2);

        // Drehachse anpassen
        // n' = qr*n*qr^-1

        quaternion n_achse; // Um einen Vektor mit einem Quaternion zu multiplizieren wird der
Vektor als ein Quaternion mit w=0 ausgedrückt
        n_achse.w = 0;
        n_achse.x = n.x;
        n_achse.y = n.y;
        n_achse.z = n.z;

        quaternion qr_inv; // invertierte Version des Orientierungsquaternion
        qr_inv.w = qr.w;
        qr_inv.x = -1 * qr.x;
        qr_inv.y = -1 * qr.y;
        qr_inv.z = -1 * qr.z;

        quaternion tmp_1;
        tmp_1 = multiply_quaternion(qr,n_achse);
        //tmp_1 = multiply_quaternion(n_achse,qr);

        quaternion tmp_2;
        tmp_2 = multiply_quaternion(tmp_1,qr_inv);
        //tmp_2 = multiply_quaternion(qr_inv,tmp_1);

        // Quaternion berechnen
        q.w = cos(lambda/2);
        q.x = sin(lambda/2)*tmp_2.x;
        q.y = sin(lambda/2)*tmp_2.y;
        q.z = sin(lambda/2)*tmp_2.z;

        //// insert log entry
        //std::string log_entry;
        //std::ostringstream temp;
        //temp << fixed;
        //temp << setprecision(6) << lat1 << "," << lon1 << "," << lat2 << "," << lon2 << ",";
        //temp << setprecision(6) <<q.w << "," << q.x << "," << q.y << "," << q.z;
        //log_entry = temp.str();
        //write_to_logfile ("beta_quaternion_log.csv",log_entry);

        return q;
}

// Multiplikation zweier Quaternions
quaternion multiply_quaternion (quaternion q1, quaternion q2)
{
        float w1 = q1.w;
        float x1 = q1.x;
        float y1 = q1.y;
        float z1 = q1.z;
        float w2 = q2.w;
        float x2 = q2.x;
        float y2 = q2.y;
        float z2 = q2.z;

        quaternion rot;

        rot.w = w1*w2 - x1*x2 - y1*y2 - z1*z2;
        rot.x = w1*x2 + x1*w2 + y1*z2 - z1*y2;
        rot.y = w1*y2 - x1*z2 + y1*w2 + z1*x2;
        rot.z = w1*z2 + x1*y2 - y1*x2 + z1*w2;

        return rot;
}

// Quaternion invertieren
quaternion invert_quaternion (quaternion q)
{
```

```
        quaternion inv;

        inv.w = q.w;
        inv.x = -1 * q.x;
        inv.y = -1 * q.y;
        inv.z = -1 * q.z;

        return inv;
}

// Rotations_Quaternion in eine 3x3 Matrix umrechnen
matrix_3x3 quat_2_matr (quaternion q)
{
        matrix_3x3 m;

        m.a1 = 1 - 2 * pow(q.y,2) - 2 * pow(q.z,2);
        m.a2 = 2 * q.x * q.y - 2 * q.w * q.z;
        m.a3 = 2 * q.x * q.z + 2 * q.w * q.y;
        m.b1 = 2 * q.x * q.y + 2 * q.w * q.z;
        m.b2 = 1 - 2 * pow(q.x,2) - 2 * pow(q.z,2);
        //m.b3 = 2 * q.y * q.z + 2 * q.w * q.x;
        m.b3 = 2 * q.y * q.z - 2 * q.w * q.x;//
        m.c1 = 2 * q.x * q.z - 2 * q.w * q.y;
        //m.c2 = 2 * q.y * q.z - 2 * q.w * q.x;
        m.c2 = 2 * q.y * q.z + 2 * q.w * q.x;//
        m.c3 = 1 - 2 * pow(q.x,2) - 2 * pow(q.y,2);

        /*
        m.a1 = pow(q.w,2)+pow(q.x,2)-pow(q.y,2)-pow(q.z,2);
        m.a2 = 2*q.x*q.y + 2*q.w*q.z;
        m.a3 = 2*q.x*q.z - 2*q.y*q.w;
        m.b1 = 2*q.x*q.y - 2*q.w*q.z;
        m.b2 = pow(q.w,2)-pow(q.x,2)+pow(q.y,2)-pow(q.z,2);
        m.b3 = 2*q.y*q.z + 2*q.w*q.x;
        m.c1 = 2*q.x*q.z + 2*q.w*q.y;
        m.c2 = 2*q.y*q.z - 2*q.w*q.x;
        m.c3 = pow(q.w,2)-pow(q.x,2)-pow(q.y,2)+pow(q.z,2);
        */

        return m;

}

// Vektor mir einer 3x3 Matrix multiplizieren
xyz_vektor vek_x_matr (xyz_vektor v, matrix_3x3 m)
{
        xyz_vektor ret;
        ret.x = v.x*m.a1 + v.y*m.a2 + v.z*m.a3;
        ret.y = v.x*m.b1 + v.y*m.b2 + v.z*m.b3;
        ret.z = v.x*m.c1 + v.y*m.c2 + v.z*m.c3;
        //ret.x = v.x * m.a1 + v.y * m.b1 + v.z * m.c1;
        //ret.x = v.x * m.a2 + v.y * m.b2 + v.z * m.c2;
        //ret.x = v.x * m.a3 + v.y * m.b3 + v.z * m.c3;

        return ret;
}

// Rotation zur alten hinzufügen
quaternion rot2 (double lat1, double lon1, double lat2, double lon2, quaternion qr)
{

        /*
        // Lat/Long in theta/phi umrechen
        theta1 = rad(90 - lat1);
        theta2 = rad(90 - lat2);

        phi1 = (lon1>180) ? rad(lon1 - 360) : rad(lon1);
        phi2 = (lon2>180) ? rad(lon2 - 360) : rad(lon2);


        // x,y,z Vektoren aus den Länge/Breite Punkten berechnen
        //v1.x = sin(theta1)*cos(phi1);
        //v1.y = sin(theta1)*sin(phi1);
        //v1.z = cos(theta1);
        //
```

```
        //v2.x = sin(theta2)*cos(phi2);
        //v2.y = sin(theta2)*sin(phi2);
        //v2.z = cos(theta2);

        // Anpassung an das OGRE Koordinatensystem
        // ogre koordinatensystem == mein koordinatensystem
        // z == -x
        // x == -y
        // y == z
        v1.z = -1 * sin(theta1)*cos(phi1);
        v1.x = -1 * sin(theta1)*sin(phi1);
        v1.y = cos(theta1);

        v2.z = -1 * sin(theta2)*cos(phi2);
        v2.x = -1 * sin(theta2)*sin(phi2);
        v2.y = cos(theta2);


        // lambda bestimmen
        lambda = vektoren_winkel(v1,v2);


        // Drehachse
        n = normalenvektor(v1,v2);
        */
        // Variablen
        double theta1, phi1, theta2, phi2 = 0;   // Drehwinkel für die Berechnung der x,y,z
Koordinaten der Winkel
        xyz_vektor v1,v2;                                         // Vektoren (0,0)->P1|P2
        xyz_vektor n;                                             // Normaleneinheitsvektor
zwischen v1, v2
        double lambda = 0;                                        // Winkel zwischen den
beiden Vektoren
        quaternion q;                                             // zu berechnendes
Quaternion
        quaternion tmp,tmp2;                                              // temporaeresa
Quaternion
        matrix_3x3 m;                                             // Rotationsmatrix um den
Vektor ins aktuelle lokale OGRE System zu bringen
        xyz_vektor org;                                           // DEBUG




        // Lat/Long in theta/phi umrechen
        theta1 = rad(90 - lat1);
        theta2 = rad(90 - lat2);

        phi1 = (lon1>180) ? rad(lon1 - 360) : rad(lon1);
        phi2 = (lon2>180) ? rad(lon2 - 360) : rad(lon2);


        v1.z = -1 * sin(theta1)*cos(phi1);
        v1.x = -1 * sin(theta1)*sin(phi1);
        v1.y = cos(theta1);

        v2.z = -1 * sin(theta2)*cos(phi2);
        v2.x = -1 * sin(theta2)*sin(phi2);
        v2.y = cos(theta2);

        //DEBUG
        //cout << "v1 : " << v1.x << " " << v1.y << " " << v1.z << endl;
        //cout << "v2 : " << v2.x << " " << v2.y << " " << v2.z << endl;

        // lambda bestimmen
        lambda = vektoren_winkel(v1,v2);

        // Drehachse
        n = normalenvektor(v1,v2);
        org.x = n.x;
        org.y = n.y;
        org.z = n.z;

        //DEBUG
        //cout << "org: " << org.x << " " << org.y << " " << org.z << endl;
```

```
        //Drehachse ins lokale ogresystem umrechnen

        // rotationsmatrix aus dem invertierten ROT Quaternion berechnen
        tmp.w = qr.w;
        tmp.x = -1*qr.x;
        tmp.y = -1*qr.y;
        tmp.z = -1*qr.z;

        m = quat_2_matr(tmp);

        //DEBUG
        //cout << "m1 : " << m.a1 << " " << m.a2 << " " << m.a3 << endl;
        //cout << "m2 : " << m.b1 << " " << m.b2 << " " << m.b3 << endl;
        //cout << "m3 : " << m.c1 << " " << m.c2 << " " << m.c3 << endl;

        // globale achse(vektor) mit der rotationsmatrix multiplizieren => lokale achse
        n = vek_x_matr(n,m);

        //DEBUG
        //cout << "n  : " << n.x << " " << n.y << " " << n.z << endl;


        //aus der lokalen achse und dem winkel ein quaternion NEU erstellen

        // Quaternion berechnen
        tmp2.w = cos(lambda/2);
        tmp2.x = sin(lambda/2)*n.x;
        tmp2.y = sin(lambda/2)*n.y;
        tmp2.z = sin(lambda/2)*n.z;

        //5. ROT = ROT*NEU

        //q = multiply_quaternion (qr,tmp2);


        //qr->w = q.w;
        //qr->x = q.x;
        //qr->y = q.y;
        //qr->z = q.z;

        //DEBUG
        //cout << "global inpoint: " << org.x << " " << org.y << " " << org.z << "   ";// << endl;
        //cout << "lokal outpoint: " << n.x << " " << n.y << " " << n.z << endl;
        //cout << "old rot;      ; " << tmp.w << " " << -1*tmp.x << " " << -1*tmp.y << " " << -1*q.z
<< endl;
        return tmp2;

}
```

QUELLTEXT: KREIS_BERECHNUNGEN.H

```
#pragma once
#include <stdio.h>


class zentrum
{
	public:
		zentrum();
		~zentrum();
		void calculate();
		void print();
		void reset();
		int nr;
		double radsum;
		double radius;
		double xsum;
		double ysum;
		double x;
		double y;

};

class line
{
	public:
		line();
		~line();
		int x1;
		int x2;
		int y1;
		int y2;
};

struct mGerade
{
    double m;
    double b;
};


struct mPunkt
{
	double x;
	double y;
};

struct mKreis
{
	struct mPunkt mittelpunkt;
	double radius;
};

// steigung und y abschnitt einer gerade durch 2 punkte
mGerade gerade(double x1, double y1, double x2, double y2);

// mittelpunkt einer strecke durch 2 punkte
mPunkt mittelpunkt(double x1,double y1,double x2, double y2);

//schnittpunkt zweier geraden
mPunkt schnittpunkt (mGerade g1, mGerade g2);

//normale einer geraden durch einen punkt
mGerade normale (mGerade g, mPunkt p);

double euklidische_distanz (mPunkt p1, mPunkt p2);

// mittelpunkt und radius eines kreises anhand 3er punkte
```

```
mKreis _3punktkreis (mPunkt p1, mPunkt p2, mPunkt p3);
```

QUELLTEXT: KREIS_BERECHNUNGEN.CPP

```
#pragma once
#include "kreis_berechnungen.h"


#include <math.h>

void zentrum::reset()
{
        this->nr = 0;
        this->radsum = 0;
        this->radius = 0;
        this->xsum = 0;
        this->ysum = 0;
        this->x = 0;
        this->y = 0;
}

void zentrum::calculate()
{
        this->x = this->xsum/this->nr;
        this->y = this->ysum/this->nr;
        this->radius = this->radsum/this->nr;
}

void zentrum::print()
{
        printf ("new averaged centre: %.1f | %.1f\tradius:%.1f\n", this->x, this->y, this->radius);
}

zentrum::zentrum()
{
        this->xsum = 0;
        this->ysum = 0;
        this->x = 0;
        this->y = 0;
        this->nr = 0;
        this->radsum = 0;
        this->radius = 0;
}

zentrum::~zentrum()
{
}


line::line()
{
        this->x1 = 0;
        this->y1 = 0;
        this->x2 = 0;
        this->y2 = 0;
}
line::~line()
{
}


// steigung und y abschnitt einer gerade durch 2 punkte
mGerade gerade(double x1, double y1, double x2, double y2)
{
        double m = (y1+(y2*-1)) / (x1 + (-1*x2));
        double b = y2 - (x1*m);
        struct mGerade rv = {m,b};
        return rv;
}

// mittelpunkt einer strecke durch 2 punkte
```

```
mPunkt mittelpunkt(double x1,double y1,double x2, double y2)
{
        double x = (x1+x2)/2;
        double y = (y1+y2)/2;
        struct mPunkt rv = {x,y};
        return rv;
}

//schnittpunkt zweier geraden
mPunkt schnittpunkt (mGerade g1, mGerade g2)
{
        double x = (g2.b-g1.b)/(g1.m-g2.m);
        double y = g1.m*x+g1.b;
        struct mPunkt punkt = {x,y};
        return punkt;
}

//normale einer geraden durch einen punkt
mGerade normale (mGerade g, mPunkt p)
{
        double m = -1 / g.m;
        double b = p.y - (m*p.x);
        struct mGerade rv = {m,b};
        return rv;
}


double euklidische_distanz (mPunkt p1, mPunkt p2)
{
        double dist = sqrt(pow((p1.x - p2.x),2) + pow((p1.y-p2.y),2));
        return dist;
}

// mittelpunkt und radius eines kreises anhand 3er punkte
mKreis _3punktkreis (mPunkt p1, mPunkt p2, mPunkt p3)
{

        //gerade zu p1p2
        struct mGerade p1p2;
        p1p2 = gerade(p1.x,p1.y,p2.x,p2.y);
        struct mPunkt mp1p2;
        mp1p2 = mittelpunkt(p1.x,p1.y,p2.x,p2.y);
        struct mGerade np1p2;
        np1p2 = normale(p1p2,mp1p2);


        //gerade zu p2p3
        struct mGerade p2p3;
        p2p3 = gerade(p2.x,p2.y,p3.x,p3.y);
        struct mPunkt mp2p3;
        mp2p3 = mittelpunkt(p2.x,p2.y,p3.x,p3.y);
        struct mGerade np2p3;
        np2p3 = normale(p2p3,mp2p3);


        struct mPunkt mitte = schnittpunkt(np1p2,np2p3);

        //radius
        double radius = euklidische_distanz(mitte,p1);

        //printf ("radius: %f\n", radius);

        struct mKreis rv = {mitte,radius};

        return rv;

}
```

QUELLTEXT: REFERENZGITTER.H

```
#pragma once


#include <vector>
#ifndef _KALIB_
#include "kalib_functions_and_types.h"
#endif


using namespace std;




#ifndef REFERENZGITTER_C
#define REFERENZGITTER_C

struct closest_refpoints
{
	double near_lat;
	double far_lat;
	double near_long;
	double far_long;
};

class Referenzgitter
{
public:
	Referenzgitter(void);
	~Referenzgitter(void);

	vector<vector <passpunkt> > points;
	sorted_ll_list* sort;
	vector<Referenzpunkt>* ref_points;
	zentrum center;

	void init(mPunkt zentrum, sorted_ll_list* sort);

	// get lat and long values for the closest ref_points to a given point
	closest_refpoints closest_lat_lon(mPunkt p);

	//claculate globe lat and long for a given point
	vector<double> calc_lat_long (passpunkt &in_pp);
};

#endif
```

QUELLTEXT: REFERENZGITTER.CPP

```
#pragma once
#include "Referenzpunkt.h"


Referenzpunkt::Referenzpunkt(void)
{
}


Referenzpunkt::~Referenzpunkt(void)
{
}
```

QUELLTEXT: MY_UBCORE.H

```
#pragma once
#ifndef MYUBCORE
#define MYUBCORE



#include "stdafx.h"



int get_frame (FIREi_CAMERA_FRAME *CameraFrame, FIREi_ISOCH_ENGINE_HANDLE *g_hIsochEngine, int
timeout, IplImage *frame_out)
{
        FIREi_STATUS FiStatus;
        FiStatus = FiGetNextCompleteFrame(CameraFrame, *g_hIsochEngine, timeout);
        if ( FiStatus != FIREi_STATUS_SUCCESS )
        {
                //printf(" FiGetNextCompleteFrame failed with status code %x\n", FiStatus);
                return 1;
        }
        frame_out->imageData = (char*) CameraFrame->pCameraFrameBuffer;
        return 0;
}



/*============================================================================
KALIBRIERUNGSROUTINE ZUR EINSTELLUNGEN DES KAMERABILDES

- aufnehmen von x frames
- jeden fram equalizen, smoothen,.....
- maximum werte kummulieren

- evtl ein opencv fenster öffnen und schließen
- am schluß einen threshold bestimmen


- das ergebniss ist ein bild mit den "blinden flecken", die vom livebild der
  kamera abgezogen werden
============================================================================*/


int blind_spot_kalib (FIREi_CAMERA_FRAME *CameraFrame, FIREi_ISOCH_ENGINE_HANDLE *g_hIsochEngine,
int timeout, IplImage *frame_out)
{
        cvNamedWindow( "kalib", CV_WINDOW_AUTOSIZE);
        cvMoveWindow("kalib",810,350);
        //cvNamedWindow("histogram",1);
        int kalib_thres = 0;
        cvCreateTrackbar("thres","kalib",&kalib_thres,255,0);

        //CvSize size = cvSize(800,600);
        CvSize size = cvSize(1024,768);

        int depth = IPL_DEPTH_8U;
        int channels = 1;
        bool quit_kalib = false;

        IplImage *kalib_frame = cvCreateImage(size,IPL_DEPTH_8U,channels);
        IplImage *actual_frame = cvCreateImage(size,IPL_DEPTH_8U,channels);
        IplImage *out_frame = cvCreateImage(size,IPL_DEPTH_8U,channels);

        //prepare histogram structure
        IplImage* image= 0;
        IplImage* imgHistogram = 0;
        IplImage* gray= 0;
```

```
        int mhist_param = 1;

        //size of the histogram -1D histogram
    int bins = 256;
    int hsize[] = {bins};

    //max and min value of the histogram
    float max_value = 0, min_value = 0;

    //value and normalized value
    float value;
    int normalized;

    //ranges - grayscale 0 to 256
    float xranges[] = { 0, 256 };
    float* ranges[] = { xranges };



        int first_frames = 0;
        while (first_frames < 60)
                {
                        //Frame holen
                        int frame_result = get_frame(CameraFrame, g_hIsochEngine,timeout,
actual_frame);

                        //processing ?!?
                        cvEqualizeHist(actual_frame,actual_frame);

                        //auf kalibrierungsframe abbilden
                        cvMax(kalib_frame,actual_frame,kalib_frame);

                        //counter
                        first_frames += 1;
                }

        //while (!quit_kalib)
        //{
                //cvCopy(kalib_frame,out_frame);
                //cvSmooth(out_frame,out_frame,CV_GAUSSIAN,7);
                //cvThreshold(out_frame,out_frame,255,kalib_thres,CV_THRESH_BINARY);


            //get the histogram and some info about it

                //planes to obtain the histogram, in this case just one
                //IplImage* planes[] = { kalib_frame };


                float steps[256] ={};
                for (int i = 0; i < 256; i++)
                {
                        cvCopy(kalib_frame,out_frame);
                        cvThreshold(out_frame,out_frame,i,255,CV_THRESH_BINARY);
                        cvSmooth(out_frame,out_frame,CV_GAUSSIAN,7);
                        CvHistogram* hist;
                        hist = cvCreateHist( 1, hsize, CV_HIST_ARRAY, ranges,1);
                        cvCalcHist(&out_frame, hist, 0, NULL);
                        cvNormalizeHist(hist, 100);
                        cvCalcHist(&out_frame, hist, 0, NULL);
                        cvNormalizeHist(hist, 100);
                        steps[i] = cvQueryHistValue_1D( hist, 255);
                        cvReleaseHist(&hist);
                        cvShowImage("kalib",out_frame);
                        //printf ("step: %d, value: %f\n",i,steps[i]);
                        printf(".");
                        //cvWaitKey(5);
                }

                int step_nr = 0;
        #pragma once
#ifndef MYUBCORE
#define MYUBCORE
```

L

```
#include "stdafx.h"



int get_frame (FIREi_CAMERA_FRAME *CameraFrame, FIREi_ISOCH_ENGINE_HANDLE *g_hIsochEngine, int
timeout, IplImage *frame_out)
{
        FIREi_STATUS FiStatus;
        FiStatus = FiGetNextCompleteFrame(CameraFrame, *g_hIsochEngine, timeout);
        if ( FiStatus != FIREi_STATUS_SUCCESS )
        {
                //printf(" FiGetNextCompleteFrame failed with status code %x\n", FiStatus);
                return 1;
        }
        frame_out->imageData = (char*) CameraFrame->pCameraFrameBuffer;
        return 0;
}



/*=============================================================================
KALIBRIERUNGSROUTINE ZUR EINSTELLUNGEN DES KAMERABILDES

- aufnehmen von x frames
- jeden fram equalizen, smoothen,.....
- maximum werte kummulieren

- evtl ein opencv fenster öffnen und schließen
- am schluß einen threshold bestimmen


- das ergebniss ist ein bild mit den "blinden flecken", die vom livebild der
  kamera abgezogen werden
=============================================================================*/


int blind_spot_kalib (FIREi_CAMERA_FRAME *CameraFrame, FIREi_ISOCH_ENGINE_HANDLE *g_hIsochEngine,
int timeout, IplImage *frame_out)
{
        cvNamedWindow( "kalib", CV_WINDOW_AUTOSIZE);
        cvMoveWindow("kalib",810,350);
        //cvNamedWindow("histogram",1);
        int kalib_thres = 0;
        cvCreateTrackbar("thres","kalib",&kalib_thres,255,0);

        //CvSize size = cvSize(800,600);
        CvSize size = cvSize(1024,768);

        int depth = IPL_DEPTH_8U;
        int channels = 1;
        bool quit_kalib = false;

        IplImage *kalib_frame = cvCreateImage(size,IPL_DEPTH_8U,channels);
        IplImage *actual_frame = cvCreateImage(size,IPL_DEPTH_8U,channels);
        IplImage *out_frame = cvCreateImage(size,IPL_DEPTH_8U,channels);

        //prepare histogram structure
        IplImage* image= 0;
        IplImage* imgHistogram = 0;
        IplImage* gray= 0;



        int mhist_param = 1;

        //size of the histogram -1D histogram
    int bins = 256;
    int hsize[] = {bins};

    //max and min value of the histogram
    float max_value = 0, min_value = 0;
```

```
    //value and normalized value
    float value;
    int normalized;

    //ranges - grayscale 0 to 256
    float xranges[] = { 0, 256 };
    float* ranges[] = { xranges };



        int first_frames = 0;
        while (first_frames < 60)
                {
                        //Frame holen
                        int frame_result = get_frame(CameraFrame, g_hIsochEngine,timeout,
actual_frame);

                        //processing ?!?
                        cvEqualizeHist(actual_frame,actual_frame);

                        //auf kalibrierungsframe abbilden
                        cvMax(kalib_frame,actual_frame,kalib_frame);

                        //counter
                        first_frames += 1;
                }

        //while (!quit_kalib)
        //{
                //cvCopy(kalib_frame,out_frame);
                //cvSmooth(out_frame,out_frame,CV_GAUSSIAN,7);
                //cvThreshold(out_frame,out_frame,255,kalib_thres,CV_THRESH_BINARY);


            //get the histogram and some info about it

                //planes to obtain the histogram, in this case just one
                //IplImage* planes[] = { kalib_frame };


                float steps[256] ={};
                for (int i = 0; i < 256; i++)
                {
                        cvCopy(kalib_frame,out_frame);
                        cvThreshold(out_frame,out_frame,i,255,CV_THRESH_BINARY);
                        cvSmooth(out_frame,out_frame,CV_GAUSSIAN,7);
                        CvHistogram* hist;
                        hist = cvCreateHist( 1, hsize, CV_HIST_ARRAY, ranges,1);
                        cvCalcHist(&out_frame, hist, 0, NULL);
                        cvNormalizeHist(hist, 100);
                        cvCalcHist(&out_frame, hist, 0, NULL);
                        cvNormalizeHist(hist, 100);
                        steps[i] = cvQueryHistValue_1D( hist, 255);
                        cvReleaseHist(&hist);
                        cvShowImage("kalib",out_frame);
                        //printf ("step: %d, value: %f\n",i,steps[i]);
                        printf(".");
                        //cvWaitKey(5);
                }

                int step_nr = 0;
                float step_dif = 0;

                for (int i=0;i<254;i++)
                {
                        float dif = abs(steps[i] - steps[i+1]);
                        if (dif > step_dif)
                        {
                                step_nr = i;
                                step_dif = dif;
                        }

                }
                step_nr += 1;
                printf ("\nbest step:%d\n",step_nr);
```

```
		cvCopy(kalib_frame,out_frame);
		cvThreshold(out_frame,out_frame,step_nr,255,CV_THRESH_BINARY);
		cvSmooth(out_frame,out_frame,CV_GAUSSIAN,7);
		cvShowImage("kalib",out_frame);
		cvWaitKey();

		//printf ("256: %f\n", cvQueryHistValue_1D( hist, 255));


		//cvGetMinMaxHistValue( hist, &min_value, &max_value);
		//printf("min: %f, max: %f, %i:%f\n", mhist_param,min_value,
max_value,cvQueryHistValue_1D( hist, mhist_param));
		/*
		for (int i=0;i<256;i++)
		{
			if ((cvQueryHistValue_1D( hist, i)) != 0)
			{
				printf ("value for %d:%f\n",i,cvQueryHistValue_1D( hist, i));
			}
		}*/
		//create an 8 bits single channel image to hold the histogram
		//paint it white
		//imgHistogram = cvCreateImage(cvSize(bins, 250),8,1);
		//cvRectangle(imgHistogram, cvPoint(0,0), cvPoint(256,50), CV_RGB(255,255,255),-1);

		//draw the histogram :P
		/*for(int i=0; i < bins; i++){
				value = cvQueryHistValue_1D( hist, i);
				normalized = cvRound(value*250/max_value);
				cvLine(imgHistogram,cvPoint(i,250), cvPoint(i,250-normalized),
CV_RGB(0,0,0));
		}

		cvShowImage("histogram",imgHistogram);*/
		//cvReleaseImage(&imgHistogram);

		/*switch (cvWaitKey())
			{
				case 'q':
					quit_kalib = true;
					break;
			}*/
	//}


//========create the 1D histogram==================================
//=================================================================
	//TODO automatisches anpassen des threshold wertes
	// alle werte ausprobieren ud histogramm bilden
	// beim größten sprung die schwelle setzen







	//cvDestroyWindow("kalib");
	//cvDestroyWindow("histogram");
	cvCopy(out_frame,frame_out);
	printf ("done calibrating\n");
	cvReleaseImage(&kalib_frame);
	cvReleaseImage(&out_frame);
	return 1;
}
// END KALIB ROUTINE 1 <-
//===============================



#endif	float step_dif = 0;

		for (int i=0;i<254;i++)
		{
			float dif = abs(steps[i] - steps[i+1]);
```

```
                        if (dif > step_dif)
                        {
                                step_nr = i;
                                step_dif = dif;
                        }

                }
                step_nr += 1;
                printf ("\nbest step:%d\n",step_nr);
                cvCopy(kalib_frame,out_frame);
                cvThreshold(out_frame,out_frame,step_nr,255,CV_THRESH_BINARY);
                cvSmooth(out_frame,out_frame,CV_GAUSSIAN,7);
                cvShowImage("kalib",out_frame);
                cvWaitKey();

                //printf ("256: %f\n", cvQueryHistValue_1D( hist, 255));

#pragma once
#ifndef MYUBCORE
#define MYUBCORE



#include "stdafx.h"



int get_frame (FIREi_CAMERA_FRAME *CameraFrame, FIREi_ISOCH_ENGINE_HANDLE *g_hIsochEngine, int
timeout, IplImage *frame_out)
{
        FIREi_STATUS FiStatus;
        FiStatus = FiGetNextCompleteFrame(CameraFrame, *g_hIsochEngine, timeout);
        if ( FiStatus != FIREi_STATUS_SUCCESS )
        {
                //printf(" FiGetNextCompleteFrame failed with status code %x\n", FiStatus);
                return 1;
        }
        frame_out->imageData = (char*) CameraFrame->pCameraFrameBuffer;
        return 0;
}



/*==========================================================================
KALIBRIERUNGSROUTINE ZUR EINSTELLUNGEN DES KAMERABILDES

- aufnehmen von x frames
- jeden fram equalizen, smoothen,.....
- maximum werte kummulieren

- evtl ein opencv fenster öffnen und schließen
- am schluß einen threshold bestimmen


- das ergebniss ist ein bild mit den "blinden flecken", die vom livebild der
  kamera abgezogen werden
==========================================================================*/


int blind_spot_kalib (FIREi_CAMERA_FRAME *CameraFrame, FIREi_ISOCH_ENGINE_HANDLE *g_hIsochEngine,
int timeout, IplImage *frame_out)
{
        cvNamedWindow( "kalib", CV_WINDOW_AUTOSIZE);
        cvMoveWindow("kalib",810,350);
        //cvNamedWindow("histogram",1);
        int kalib_thres = 0;
        cvCreateTrackbar("thres","kalib",&kalib_thres,255,0);

        //CvSize size = cvSize(800,600);
        CvSize size = cvSize(1024,768);

        int depth = IPL_DEPTH_8U;
        int channels = 1;
        bool quit_kalib = false;
```

```
        IplImage *kalib_frame = cvCreateImage(size,IPL_DEPTH_8U,channels);
        IplImage *actual_frame = cvCreateImage(size,IPL_DEPTH_8U,channels);
        IplImage *out_frame = cvCreateImage(size,IPL_DEPTH_8U,channels);

        //prepare histogram structure
        IplImage* image= 0;
        IplImage* imgHistogram = 0;
        IplImage* gray= 0;



        int mhist_param = 1;

        //size of the histogram -1D histogram
    int bins = 256;
    int hsize[] = {bins};

    //max and min value of the histogram
    float max_value = 0, min_value = 0;

    //value and normalized value
    float value;
    int normalized;

    //ranges - grayscale 0 to 256
    float xranges[] = { 0, 256 };
    float* ranges[] = { xranges };



        int first_frames = 0;
        while (first_frames < 60)
                {
                        //Frame holen
                        int frame_result = get_frame(CameraFrame, g_hIsochEngine,timeout,
actual_frame);

                        //processing ?!?
                        cvEqualizeHist(actual_frame,actual_frame);

                        //auf kalibrierungsframe abbilden
                        cvMax(kalib_frame,actual_frame,kalib_frame);

                        //counter
                        first_frames += 1;
                }

        //while (!quit_kalib)
        //{
                //cvCopy(kalib_frame,out_frame);
                //cvSmooth(out_frame,out_frame,CV_GAUSSIAN,7);
                //cvThreshold(out_frame,out_frame,255,kalib_thres,CV_THRESH_BINARY);


            //get the histogram and some info about it

                //planes to obtain the histogram, in this case just one
                //IplImage* planes[] = { kalib_frame };


                float steps[256] ={};
                for (int i = 0; i < 256; i++)
                {
                        cvCopy(kalib_frame,out_frame);
                        cvThreshold(out_frame,out_frame,i,255,CV_THRESH_BINARY);
                        cvSmooth(out_frame,out_frame,CV_GAUSSIAN,7);
                        CvHistogram* hist;
                        hist = cvCreateHist( 1, hsize, CV_HIST_ARRAY, ranges,1);
                        cvCalcHist(&out_frame, hist, 0, NULL);
                        cvNormalizeHist(hist, 100);
                        cvCalcHist(&out_frame, hist, 0, NULL);
                        cvNormalizeHist(hist, 100);
                        steps[i] = cvQueryHistValue_1D( hist, 255);
                        cvReleaseHist(&hist);
                        cvShowImage("kalib",out_frame);
```

```
			//printf ("step: %d, value: %f\n",i,steps[i]);
			printf(".");
			//cvWaitKey(5);
		}

		int step_nr = 0;
		float step_dif = 0;

		for (int i=0;i<254;i++)
		{
			float dif = abs(steps[i] - steps[i+1]);
			if (dif > step_dif)
			{
				step_nr = i;
				step_dif = dif;
			}

		}
		step_nr += 1;
		printf ("\nbest step:%d\n",step_nr);
		cvCopy(kalib_frame,out_frame);
		cvThreshold(out_frame,out_frame,step_nr,255,CV_THRESH_BINARY);
		cvSmooth(out_frame,out_frame,CV_GAUSSIAN,7);
		cvShowImage("kalib",out_frame);
		cvWaitKey();

		//printf ("256: %f\n", cvQueryHistValue_1D( hist, 255));


		//cvGetMinMaxHistValue( hist, &min_value, &max_value);
		//printf("min: %f, max: %f, %i:%f\n", mhist_param,min_value,
max_value,cvQueryHistValue_1D( hist, mhist_param));
		/*
		for (int i=0;i<256;i++)
		{
			if ((cvQueryHistValue_1D( hist, i)) != 0)
			{
				printf ("value for %d:%f\n",i,cvQueryHistValue_1D( hist, i));
			}
		}*/
		//create an 8 bits single channel image to hold the histogram
		//paint it white
		//imgHistogram = cvCreateImage(cvSize(bins, 250),8,1);
		//cvRectangle(imgHistogram, cvPoint(0,0), cvPoint(256,50), CV_RGB(255,255,255),-1);

		//draw the histogram :P
		/*for(int i=0; i < bins; i++){
				value = cvQueryHistValue_1D( hist, i);
				normalized = cvRound(value*250/max_value);
				cvLine(imgHistogram,cvPoint(i,250), cvPoint(i,250-normalized),
CV_RGB(0,0,0));
		}

		cvShowImage("histogram",imgHistogram);*/
		//cvReleaseImage(&imgHistogram);

		/*switch (cvWaitKey())
			{
				case 'q':
					quit_kalib = true;
					break;
			}*/
	//}


//========create the 1D histogram=================================
//==============================================================
	//TODO automatisches anpassen des threshold wertes
	// alle werte ausprobieren ud histogramm bilden
	// beim größten sprung die schwelle setzen
```

```
        //cvDestroyWindow("kalib");
        //cvDestroyWindow("histogram");
        cvCopy(out_frame,frame_out);
        printf ("done calibrating\n");
        cvReleaseImage(&kalib_frame);
        cvReleaseImage(&out_frame);
        return 1;
}
// END KALIB ROUTINE 1 <-
//===============================

#endif
```

QUELLTEXT: REFERENZPUNKT.H

```
#pragma once
#ifndef REFERENZPUNKT_C
#define REFERENZPUNKT_C

class Referenzpunkt
{
public:
	Referenzpunkt(void);
	~Referenzpunkt(void);

	// Koordinaten im Bild, kartesisches System
	int x_pic;
	int y_pic;

	// Koordinaten im Bild, Polarsystem
	double r;
	double phi;

	// koordinaten am Globus, Länge/Breite
	double latitude;
	double longitude;

};

#endif
```

Eidesstattliche Erklärung

Ich versichere:

- dass ich die Diplomarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.
- dass ich dieses Diplomarbeitsthema bisher weder im In- noch im Ausland einer BeurteilerIn zur Begutachtung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe
- dass diese Arbeit mit der vom Begutachter beurteilten Arbeit übereinstimmt.

Wien im Oktober 2012 Michael Holzapfel

LEBENSLAUF

Persönliche Angaben:

Name:	Michael Holzapfel
Geburtsdaten:	29.12.1981, Regensburg
Familienstand:	ledig
Staatsbürgerschaft:	Deutsch

Ausbildung:

2005 -	(Fortsetzung) Diplomstudium Kartographie und Geoinformation am Institut für Geographie und Regionalforschung der Universität Wien
2002 – 2005	Diplomstudium Geographie an der Universität Regensburg
1992 – 2001	Donaugymnasium Kelheim
1988 – 1992	Grundschule Kelheimwinzer

Berufliche Tätigkeiten:

2009 -	freier Mitarbeiter der Firma Globoccess AG (Hamburg)
2008 - 2012	Studienassistent (zugeordnet Ass.-Prof. Dr. Andreas Riedl) am Institut für Geographie und Regionalforschung der Universität Wien
2006 – 2008	Tutor am Institut für Geographie und Regionalforschung der Universität Wien